Enabling modular and scalable SDA data transforms via the Raft Data Fabric

Edward Morgan
Raft
Steve Nino, Angela Sheffield
Raft

ABSTRACT

Raw SDA/SSA observational data collected for research and exploration often lack the format and datatype standardization found in operational datasets, which can lead to difficulties disseminating and using those data outside of the communities that originally collected them. Furthermore, once new datasets become of interest to operational communities, systems that perform data ingestion are usually developed on timelines much longer than necessary, causing delays and disconnects between research and operational communities. Raft, under guidance from entities in the US Air Force and Space Force, has developed a platform-based solution to mitigate these problems in a wide rangde of development and operational environments. Raft's Data Fabric enables human-driven data exploration via utilities familiar to most data scientists and developers, providing a natural visualization layer for the dissemination of new datasets. In addition, Raft's Data Fabric provides a built-in system for deploying programmatic data pipelines compatible with a wide variety of datatypes, including real-time data, relational data, and object (file) data. Raft's Data Fabric has been utilized for a use case involving the validation, transformation, and dissemination of orbital observation data collected via Space Fence. Currently, due to the wide variety of potential sub-formats and variations of source data, operators usually perform validation and transformation by hand, then manually upload the resulting data into an operational system. Research-grade data, like some cislunar observational data, is of interest in operational use cases, but may not exist in a format compatible with operational systems, meaning they must also be converted in a similarly siloed and error-prone manner. Using Raft's Data Fabric, an alternative method was developed that allowed both manually-uploaded and real-time data to be efficiently preprocessed and delivered to the appropriate system. Unless required, operations were autonomous, though the system provided step-by-step observability to confirm success. This method reduced a tedious process, taking from five minutes to an hour, to and end-to-end pipeline taking less than 30 seconds to run. In addition, a key feature of the solution is its modularity, which allows data transformations for other data types to run in the same system, or for new transformations producing different outputs using the same input.

1. INTRODUCTION

Data is at the center of the 21st century, especially in scientific fields like Space Domain Awareness (SDA). According to Space-Track.org, the preeminent source for SDA that is managed by the USSF's 18th Space Defense Squadron (SDS), well over 40,000 distinct objects (both spacecraft and debris) are currently being tracked, requiring persistent, real-time monitoring from a variety of space- and ground-based sources [15]. As the number of objects in space dramatically increases with the deployment of mass-produced CubeSats and large-scale constellations like StarLink and OneWeb, the space data problem becomes more and more difficult to manage. As this paper discusses, the twin problems of data discoverability and transformation have special significance in the SDA environment.

As the availability of data increases, the ability to discover relevant data decreases. This fact has presented a problem to scientists and engineers for years, but has increased in relevance in the last decade. With the advent of open data movements in the scientific community, newly published research usually includes dumps of relevant data used, theoretically making replication of results and expansion by other researchers easier. In a 2017 paper around data sharing in the life sciences, Culley anticipated that "data are becoming currency in science, and as such, data sets must be preserved in formats that will be useful in the future" [3]. Along with the boon of expanded access to data, several challenges have also been presented, including deciding which data to preserve, what metadata to preserve, and where to store preserved data for future discoverability. Indeed, the discoverability of high-quality data relevant to one's work has become one of the most difficult problems facing data-intensive industries today. Numerous technologies have been developed specifically to solve these problems, including Informatica, data.world, and Datahub to name a few. As will be discussed in later sections, data catalogs are especially important

when the full corpus of accessible datasets is not known a priori, requiring discovery and recommendation to uncover.

Discoverable, high-quality data is important for operational systems as well as research communities. Ustun et al. outlined the importance of data sharing and discoverability specifically for utilities in underserved communities: "The availability of accurate and reliable data has a major role to play in improving access, both in order to facilitate…and also to address the challenges associated with real-time operation." [19]. Having the right data at the right time is also key in operational use cases. Without that access, systemic issues may be more difficult to diagnose and alleviate, affecting the long-term maintainability of complex systems. In addition, operational data is often the first tool engineers and scientists look for when tasked with improving the performance, reliability, and scalability of an operational system.

Increasing the discoverability of data is typically broken down into two main components. First, aggregating discoverability information on disparate groups of datasets in a single place makes discovering data akin to visiting a supermarket. Traditionally, demarcating groups of datasets was typically achieved only in a de facto way based on the physical location of a particular dataset. In essence, a user would see datasets X, Y, and Z only because they were all stored in the same data lake, not necessarily because of any similarities or tangible significance. To combat this problem, many modern data lakes are segmented by their purpose, so all datasets associated with a particular domain (like SDA or SSA) would go into a single data lake, like the Unified Data Library [16]. This improves the data discoverability somewhat, butalso introduces two problems. First, centralization of information pertaining to a specific area siloes that information, increasing the risk of loss of access to all relevant datasets if it goes offline and lowering discoverability of the silo itself if it is not already known about. Secondly, segmenting datasets into siloes based on one set of characteristics is rarely the only way to segment them. After being segmented, it becomes significantly harder to analyze and use datasets from different siloes. This restricts the usefulness of the data writ large and contributes to bureaucratic problems across scientific and operational communities. An additional problem with this traditional model is that many data scientists, researchers, and operators don't particularly care where the data is stored beyond how difficult a dataset's location makes it to access. In other words, the physical location of a particular dataset is not a meaningful way to index or segment the relationships between datasets.

Solving this problem can involve either centralizing or federating data siloes. The modern Data Lake concept draws from the first option by moving data storage into a single, centralized, typically cloud-based repository [13]. While this does have advantages in simplifying dataset-dataset relationships inside the lake, successfully moving to a single, centralized data lake is nearly impossible in large organizations like the DoD. Instead, groups of data silos are moved into separate lakes that each try to become the canonical Data Lake, which does not solve the problem. A more modern approach to data discoverability that meets the data needs of large organizations centers on making accessing disparate sources easier. Instead of relentlessly moving data silos into data lakes as the only way to improve discoverability, federating access to data lakes allows for other systems of organizing data into relevant groupings, even including updated groupings as new ways of understanding data are developed. The problems at hand then center around providing users with a unified access pattern to each dataset, abstracting away the technical aspects of particular data lakes.

Data discoverability is also aided by the application of automatic filters and algorithms to pre-select the most relevant results to be displayed to the user. These discoverability tools can take two forms. The first is through collaborative filtering, or the process of using user-generated data to determine the quality of an item, then further refine the weighting of a particular set of user-generated data through neighbor selection [1]. Data like reviews, thumbs up/down, or usage/views numbers can be used to gather community-level preferences. These recommendations can then be further refined by the use of neighbor selection, where users of similar preferences are grouped together. A group of neighbors' preferences will then be weighted more highly than those of the broader user population, which increases the relevance of suggestions. Because of the straightforwardness of its implementation and its maturity in industry, collaborative filtering is one of the most commonly used recommendation algorithms [9].

The second group of tools used to improve a set of recommendations are typically called Content-Based Filtering (CBF) algorithms. These algorithms analyze the content of the entire corpus to be served to users, then use descriptive information about a particular user to recommend content to them [9]. Deep learning algorithms are particularly well-suited to these tasks because they can be fed large amounts of data to drive classification decisions.

CBF is generally favored when the content of a piece of data is indicative of a user's theoretical desire to discover it, as opposed to intangible characteristics that cannot be easily surmised from the corpus itself, like individual preference or taste. In the area of dataset discoverability, both approaches have been conducted at various points. Datahub, one of the most well-known open-source data catalogs, is a strong example of a CBF-based dataset recommendation tool. Users can crowd-source information about a dataset, including attaching domains, tags, glossary terms, and owners to a particular dataset [4]. This metadata surrounding a dataset can then be used to recommend the most relevant datasets for a particular user. Content-based recommendation algorithms typically take the form of deep learning algorithms, where a model ingests a corpus of content and then use it to make recommendations. The most well-known of these is the YouTube recommendation algorithm, which models recommendation as an extreme multiclass classification problem [2]. These types of recommendations can be very powerful because they can use 'implicit' statistics regarding user preferences as opposed to explicit feedback users provide. For example, YouTube uses video views as an implicit statistic to infer the popularity of a video and weights it more highly than the explicit thumbs up/down indicator users must actively provide. However, these algorithms can suffer from high performance penalties due to their use of deep neural networks instead of simpler recommendation algorithms.

Standardization (or lack thereof) of data is a common problem when working with large quantities of disparate datasets, especially when legacy sensor systems are in use. This is especially relevant to the space domain, where sensing platforms have been in operation for decades, and the feasibility of rapid iteration is sometimes low. Because of this, legacy data formats are often used for new datasets, though they may not be adequate to fully describe a particular measurement. Of specific interest in this project is the usage of the Two-Line Element (TLE) format to describe observations beyond geosynchronous orbit, especially those near the Moon. TLEs were developed to support the estimation of orbits specifically near to earth, where Earth's Keplerian Two Body Problem is applicable. A fortunate outcome of this fact is that most Low-Earth or geocentric orbits are stable, having a somewhat predictable orbit and period. Because of this, data formats that contain relatively little information, like TLEs, can be used to compute propagations. However, in cislunar space, orbital calculations must be considered in the context of the Earth-Moon Three Body Problem, which has very few stable orbits (Earth's L4 and L5 points) [7]. Holzinger, Chow, and Garretson state this fact quite plainly: "TLEs are not a useful mechanism to keep catalogs, share trajectories, or task sensors in cislunar space" [7]. Issues arise when attempting to encode a cislunar observation as a TLE, as the observational information is only valid for a short period of time and should not be used for later calculations. Due to the legacy nature of many operational systems, however, TLEs remain one of the most popular formats for processing observational measurements, and thus there is still a need to be used for initial or very early categorizations of cislunar objects.

Similar to format standardization of data, determining the lineage of a dataset is extremely important when determining fit for a research or operational purpose. Data lineage is the path of transformations a dataset underwent between its initial measurement via a sensing system and its current state. Transformations comprising a dataset's lineage may include data format changes, merges with other datasets, or filtering/down-selection, among other processes. Data lineage is valuable to users as it removes the layer of abstraction between a dataset and the underlying data. Chains of trust may be established between datasets in a system to provide backing for dataset usage in operational contexts, or lineage may be used as a tool to determine faulty processes injecting anomalous data. While data lineage has several beneficial uses, it suffers from two problems: lineage is expensive to store and expensive to query [8].

Usually, either increasing amounts of storage must be used for lineage metadata in order to support efficient querying, or query methods use increasing amounts of time to execute due to backtracking or inefficient search spaces [6]. Current methods to reduce both query complexity and storage requirements usually require some idea of the relationship between datasets in a system. If datasets are connected in a tree-like structure without many interconnections, then recursive querying can work fairly well. However, many scientific datasets contain many interconnections between datasets, where a dataset may be merged with others several times and then filtered multiple times. In these situations, it is unlikely that recursive querying will evaluate quickly, so other storage models must be explored, including interval containment graphs [5]. Regardless of the problems inherent with storing lineage data, the benefits it provides for data discovery are evident. In scientific contexts, lineage metadata can be used to improve the efficiency of analysis by aiding in the determination of a dataset's provenance [6].

2. MOTIVATION

As part of USAF SBIR #FA865021C9320, Raft partnered with the USSF 18th SDS to address key pain points in the data ingest, discovery, and upload process. Through direct user research sessions, Raft's existing Data Fabric work was expanded to improve upon the following situations.

Partnerships with research institutions are common in the SDA community. Researchers provide valuable data on emerging areas of interest to the operational community, notably including cislunar space. However, given the novelty of some new data feeds, gaining compatibility with existing processing systems is difficult without sets of transformations that may be custom for each data feed. Each individual set of transformations may change as new capabilities are developed and are rarely as static as those expected for operational use cases. Especially in early stages of integration with operational systems, reducing the delay until research data can be transformed and uploaded is very important. A pertinent example of this is with cislunar data, which is usually found in different formats than LEO or GEO data.

Data ingestion is another pain point for SDA operators. Established operational sensor data are commonly sent directly to large-scale data lakes like the Unified Data Library (UDL) for analysis and dissemination and have the benefit of standardization on ingest from sensors [16]. However, many sources of data are not directly sent to a data lake, nor do they always have rigid standardization, meaning the ingestion process can be more bespoke depending on the system doing analysis. For these datasets, a user-friendly method of uploading a dataset can dramatically increase discoverability versus keeping it siloed.

Another pain point occurs when attempting to cross-reference two datasets, one from a standardized big data lake and one from a non-standard source. In this case, any analysis or transformation will need to programmatically access the data lake as well as read the non-standard source. A system that allows for ingestion of both standard and non-standard sources in the same environment would enable easier analysis and transformation.

More broadly, there are additional technical motivations to consider when developing a system intended to support flexibility for both standard and non-standard data sources. Reliability of the system is paramount, as is the ability to track the progress of a particular dataset in a transformation pipeline.

3. SYSTEM DESIGN

The design of Raft's Data Fabric may be broken down into several main functions. First, the system must allow for deterministic data transforms that can run entirely autonomously. Towards that end, the transforms should be configured to encapsulate any preconditions needed to start processing. The system should abstract the particular medium (streaming, relational, or object/file) used for storing a dataset from the data control plane; transformations must still take that information into account. Transforms should be isolated from one another unless explicitly configured to do so, which provides cleaner lineage graphs while adhering to the principle of least concern. A core function of the open data architecture discussed in this paper involves the composability of data transforms. When a data product is generated as a result of a transform, additional transformations should be able to be developed using it without re-architecting the original transformation. As more data products are generated, their lineage should be tracked in a way that provides visibility and discoverability to both R&D and operational communities. An example set of composable transformations is shown in Fig. 1.

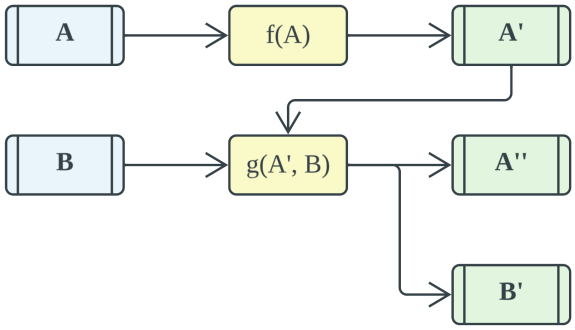

Fig. 1. Sample Two-Stage Transformation

The transformation `f(A)` may have been developed earlier, to support an initial use case. Later, a transformation `g(A', B)` may be needed to support two additional data products. No explicit reconfiguration of `f(A)` should happen, though the lineages of `A''` and `B` should reflect their provenance.

Raft's Data Fabric (RDF) was developed in accordance with these main functions. The entire stack resides on the Kubernetes deployment system, which provides reliability, fault tolerance and horizontal scaling for applications running on it. All services required to run RDF are entirely self-contained, including those used for data storage (Apache Kafka, PostgreSQL, S3-compatible MinIO), authentication and authorization (Keycloak and Open Policy Agent), data analysis and visualization (Apache Superset), in-situ development (JupyterHub, Visual Studio Code Web), and distributed querying (Trino). This means that, given appropriate resources, an instance of RDF can run on a laptop, local server, air-gapped system, or enterprise cloud; all four environments were used during the course of this project. The only technical requirements for deploying RDF, beyond appropriate RAM and CPUs for the use case, are a container runtime like Docker and a Kubernetes cluster. Every service in RDF can be interacted with and controlled via a web UI.

During the development of RDF, three data modalities were identified as sufficient to cover the vast majority of use cases: message streams, relational models, and objects/files. To that end, each medium was added to RDF and was able to be used for any dataset that requires it. To accomplish the earlier stated objective of abstracting a dataset's particular storage medium from management of the data ontology, several abstractions were created and implemented in RDF. A *Data Source* identifies a source of one or more *Dataset*s, which each uniquely identify a particular logical grouping of data in one of the three storage media. Transformations may be applied to one or more *Dataset*s and may produce zero or more *Dataset*s as output, which are re-indexed in RDF for later discovery and usage.

With many possible relationships between different *Dataset*s, and the objective stated earlier that transforms should have the ability to be autonomous, the streaming storage medium (Kafka) was given two responsibilities. First, it stored raw data as they were ingested from streaming *Data Source*s, acting as a storage medium. Additionally, it served as an Event Bus for all ingestion and transformation events in the system. When new datasets were added to RDF, a sentinel message containing descriptive information along with metadata and the dataset's location on a storage medium was published to an internal Kafka topic. Any active transformation applications would read the message, apply rules to determine if processing should occur, then use the sentinel message to perform their transformations. This processing flow is shown below in Fig. 2 for the application developed for the 18th SDS.

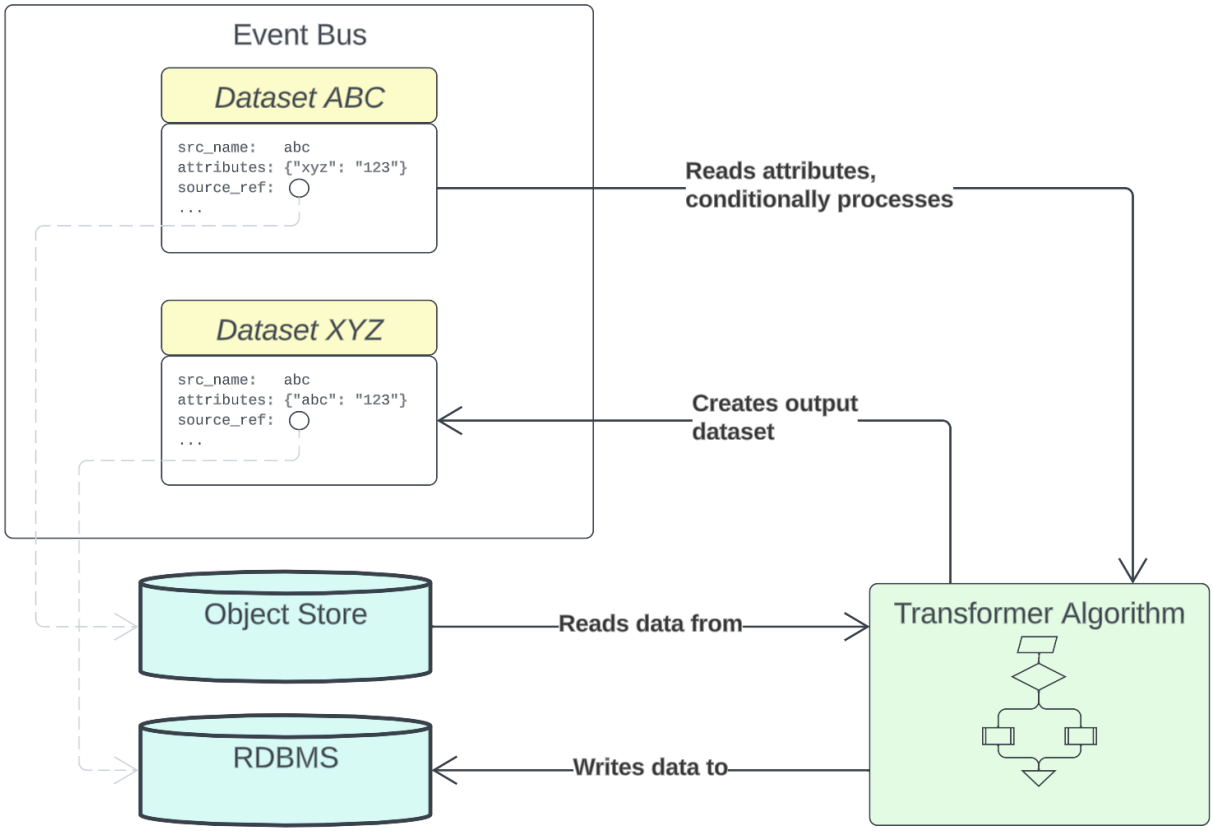

Fig. 2. Event Bus Architecture for Ingestion and Transformations

Three applications were developed for the 18th and 19th SDS to facilitate autonomous data transformation and dissemination. The first application developed ingests cislunar observation data in the Minor Planet Center Optical Observation (MPC-OO) format, converts it into an internal binary format suitable for transformation and easy dissemination to data lakes like the Unified Data Library. Data received in the MPC-OO format was research-grade observation data pertaining to artificial satellites in cislunar space. These transformations contained two parts. First, a Lark Extended Backus-Naur (EBNF)-based grammar was generated to fully represent MPC-OO messages. Next, a parser was implemented using the Lark grammar in Python to perform any required translations. Finally, a modular data uploader backend was written, also in Python, to automatically encode the translated data in a binary wire format and produce it to Apache Kafka or to S3, where it could easily be used in Jupyter notebooks or sent to external systems for further processing. A diagram showing the flow for this application is shown below in Fig. 3.

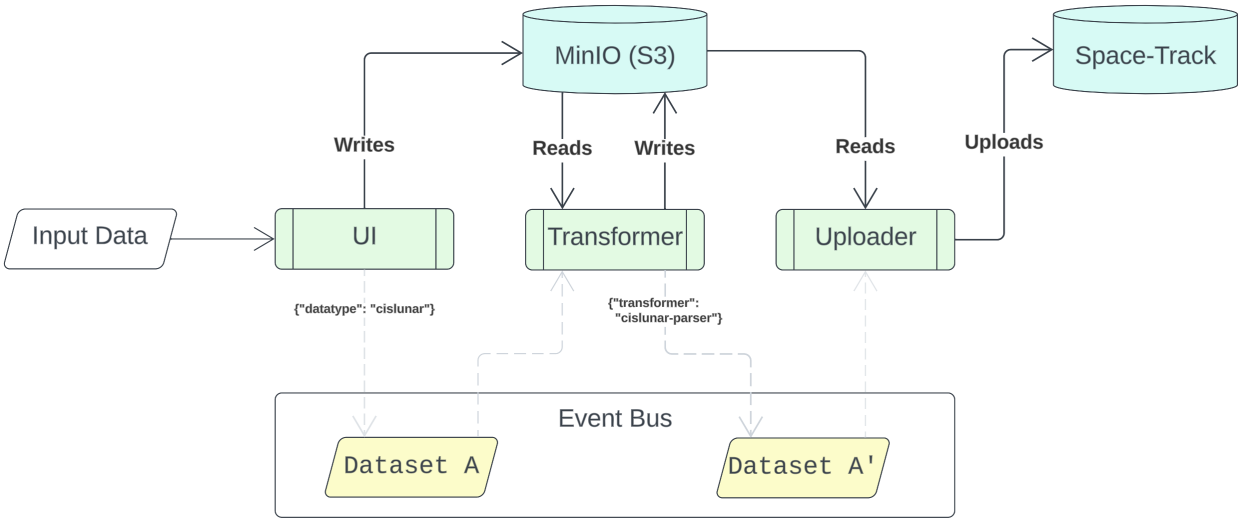

Fig. 3. Data Transformation for MPC-OO or Space-Fence Data

The second transform application was built specifically to parse and transform LEO observational data from the Space Fence surveillance system. Incoming data were encoded in a Space Fence `MetricObservationTrackMsg` XML format and two output transformations were required. First, a context-aware transformation was developed that would analyze source data to generate an output product standardized for downstream consumption, delineating observations based on whether particular measurements were present (for example, right ascension/declination or azimuth/altitude). The second transformation pursued but not deployed in this project was to standardize the orbital reference frame for all observations processed by a given transformation, in order to make configuration of downstream systems easier. This application was built in Python using the `lxml` and `pint` libraries.

The final applications developed for this project generalized operations outside the individual transformations and were compatible with both transformers described above. Using the event bus architecture described in Fig. 2, a framework was designed and implemented to provide 'hooks' to a particular data transformer, allowing a list of preconditions to drive whether the transformation was invoked.

The *transformer harness* was developed to encapsulate the process of 1) subscribing to the event bus, 2) determining whether a registered transformer should run based on preconditions configured at deployment time, 3) retrieving datasets from their storage media and preparing them for transformation, and 4) outputting the transformed dataset to the configured storage medium and registering a new *Dataset* with RDF. This application was designed to be modular with regards to the processing being done inside a transformation, such that a new transformer could be written and dropped into the harness without re-architecting it. An *uploader service* was also developed to aid in the automatic transmission of output data products to downstream systems. Using the same event bus architecture described above, a particular *uploader service* could be configured to listen for a set of preconditions and take an action when fulfilled. Both services treated specific transformations and datasets in the same way, allowing them to be abstracted away.

4. INTEGRATION

In the context of the transformations for the 18th/19th SDS, a user-based flow would be as follows. For the 18th/19th SDS use case, datasets used an S3-compatible object store (MinIO) as the storage medium. First, a researcher or data provider would upload new observation data to RDF via the built-in web UI. That process would include the provider selecting a set of attributes to be associated with the dataset. Which tags were selected would determine which transformations would be applied. Once the dataset was uploaded to RDF, it would be automatically transformed into the output format based on the tags selected. The transformed data would then be constituted as a new dataset, reflected as both a newly-created internal *Dataset* and a new object accessible in the object store via the same web UI used to upload data. In the same manner as the data transformation application, the *uploader service*

would operate on new datasets, using attributes to determine how to process them. For data uploaded for the 18th/19th SDS use case, the *uploader service* would automatically send transformed data products directly to Space-Track.org. Once on Space-Track, the data was then further disseminated to operational systems for direct use by users. Fig. 4 below shows the RDF interface at the point where a user supplies a dataset via the web UI, giving it a set of tags that will be used by downstream transformers to conditionally begin processing.

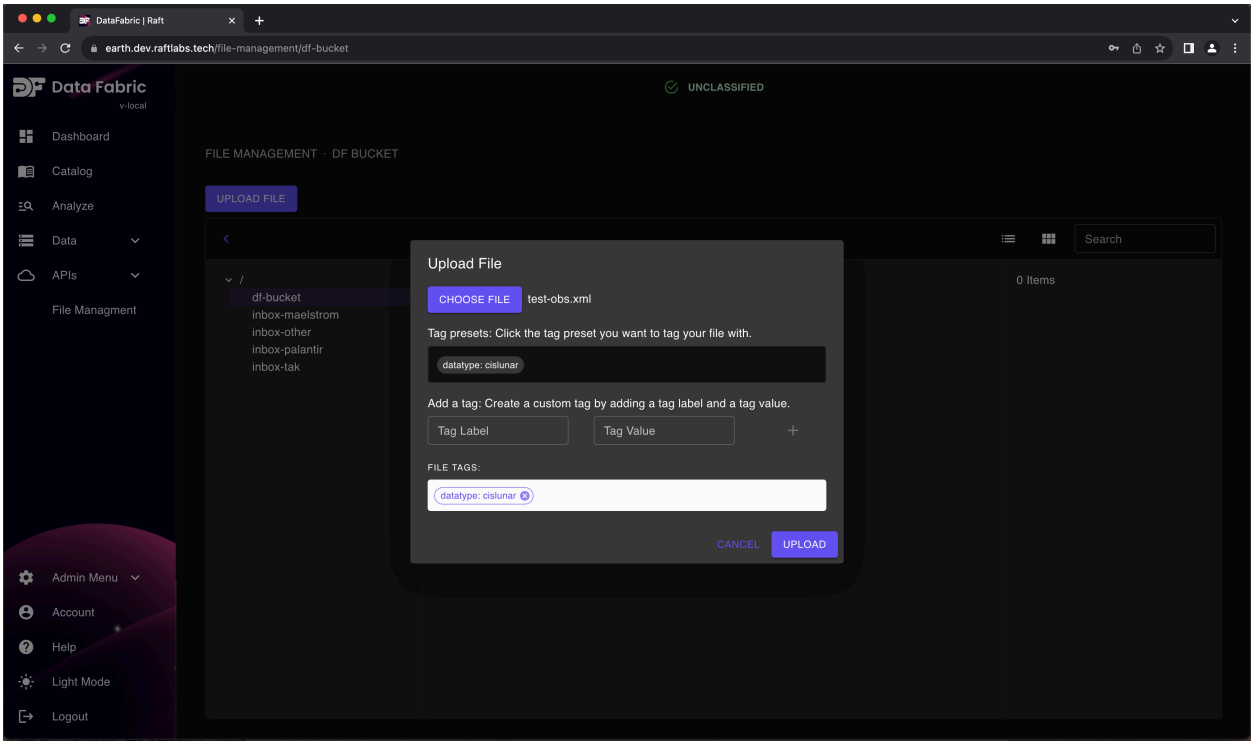

Fig. 4. Raft Data Fabric UI when a user supplies a file-based dataset

The RDF installation used was deployed on a variety of hardware configurations. For system development, RDF was installed directly on bare-metal servers. For acceptance testing and user onboarding, a cloud-based architecture was used. A Kubernetes cluster using the RKE2 distribution was deployed, with the USAF Big Bang software stack installed on top of it. This cluster was deployed inside an AWS GovCloud (IL2) account. For operational use, RDF was deployed inside the VAULT data platform in IL4. In each environment, the deployment process was identical, requiring only configuration changes to support each.

5. SYSTEM TESTING & RESULTS

Performance testing of the system components was done. All tests except end-to-end functionality were run on an Intel i7-13700k (24-core) system with 32Gb RAM running Ubuntu 22.04.3 LTS.

Component/Function	Number of runs	Mean (s)	Standard Dev. (s)	99th Percentile (s)
Obs. Transformer for 10,000 unique measurements	1000	4.55255	0.14465	4.61145
Uploader service for a 10 Mb file	30	6.63333	0.55605	7.71
End-to-End for 10,000 unique measurements	104	70.88161	115.567	680.43045

When compared against the current process of manual transformation and upload, RDF achieves a significant speedup in terms of both time and cognitive load, while also streamlining the process by which researchers can provide data via the web UI. Additional functionality not covered in these tests includes the ability to launch Jupyter

Notebooks to analyze raw and transformed datasets and the ability to improve the discoverability of these datasets via the data catalog in RDF, Datahub. With improvements to the various components, including parallelism, efficient algorithm/library selection, and vertical scaling, it is expected that this improvement over the existing method will become even more pronounced.

6. CONCLUSION

Raft's Data Fabric (RDF) has been developed as a general-purpose 'single pane of glass' over disparate data sources, be they enterprise data lakes, smaller data siloes, or one-off collections of datasets. It is composed of a technical stack that allows it to have modularity, simplistic deployment, data/metadata ingestion flexibility, and developer- and scientist-friendly abstractions for creating automatic transformations. In this project, those capabilities were extended with further development to support the 18th and 19th SDS of the U.S. Space Force, who had a use case involving automatic data transformations and propagations from disparate sources. That work has been completed over the past two years and has culminated in the deployment of RDF in various environments to support research and operational users. The results of this work, as described above, significantly reduce both the temporal and cognitive load placed on USSF operators for this particular use case. However, the nature of RDF leads to numerous opportunities for expansion.

There are two dimensions that the transformation work done in this project may be expanded: horizontally or vertically. As additional, unrelated (or possibly interrelated) use cases are found, new data transforms may be added to the system that take the same or different input products and produce different output products. There has already been significant work done at Raft to develop additional transforms specifically as they relate to the two other data storage modalities implemented: relational (RDBMS) and message streaming (Kafka). In the 2022 and 2023 USAF AFWERX Datathons, operational data sources were successfully ingested in real time from enterprise data lakes like the Unified Data Library and were used for AI/ML model development utilizing the Jupyter Notebook server in RDF. Similarly, the USAF Metadata Repository (MDR) was ingested into RDF as a relational database, allowing querying and visualization of dataset interconnections in a web UI. This is a powerful area of future work that can streamline the process of discovering, analyzing, and using real-time data in both scientific and operational capacities.

A pertinent example of future expansion using RDF is anomaly detection within one or more real-time, streaming datasets. Research and development could be conducted inside RDF using Jupyter Notebooks, leveraging the same data connections in identical environments to production use (all inside RDF). Iteration in development is easy due to the discoverability that RDF bakes into datasets and transforms. Those notebooks could be easily encoded as new transforms, then onboarded to RDF with appropriate programmatic hooks for the input sources and output data products. As long as the appropriate sources are brought into RDF via ingestion using a purpose-built library developed by Raft, anomaly detection for a wide variety of use cases could be enabled, whether for satellite trajectories, air or sea tracks, or audit trails in a software system.

A direct area of follow-on work to this project, where new logic is added to the existing transformations, can be categorized as vertical expansion. As additional steps are needed to fully transform an input dataset to an output product, they can be added to the ecosystem of transformers in RDF to create data pipelines, similar to what are found in systems like Apache Airflow or NiFi but more readily accessible in RDF due to its ease of integration. Much like adding transformations for new use cases (horizontal expansion), adding new 'links' in a data pipeline can first be developed in the Jupyter Notebooks, then tested and released as a new transform. Importantly, one unique advantage RDF has over existing data pipeline solutions is the ability to develop new transforms by hooking directly into the output product(s) generated by the production pipeline, drastically simplifying the process of development and integration. For example, standardization of observational data to a single orbital reference frame is not always possible when disseminating results across separate organizations or agencies. A trivial addition would be to add *N* parallel transforms using the output product generated by this project, where each converts the orbital reference frame and creates a new output dataset (for example, geocentric inertial, geocentric fixed, or topocentric horizon).

Combining two distinct features of RDF, attribute-based processing triggers and abstracted data storage modalities, would enable a further category of future work by enabling the cross-referencing of multiple types of data at the same time. While this is already a feature of several data tools like Trino, Presto, and Apache Flink, no other

platform contains end-to-end utilities for discovering new datasets, ingesting them, developing processing algorithms, and easily deploying them as transforms. There has been some work started at Raft around this type of transformation, namely including cross-referencing real-time streaming satellite positional TLEs from a data lake with historical positions of geosynchronous communications satellites like Iridium or Inmarsat stored in MinIO (S3). Further enhancements are envisioned around the lineage of these multi-source data transforms. Once a data product is generated, a common next step is to question the provenance of the data. Establishing a graph-based relationship between data products, from external sources ingested into the system to output products.

Overall, the work done in this project establishes a working baseline for the types of orbital transformations that can be performed on SDA data for LEO, GEO, and cislunar datasets. A flexible, module system has been designed and implemented to allow operators and researchers alike to discover data, ingest those data, develop transformation algorithms, and operationalize those algorithms in a single platform. Further work can expand both the diversity and complexity of these transforms to enable better SDA across the space community. It is the authors' hope that as the number of space-borne objects requiring tracking exponentially increases, Raft's Data Fabric will be used as a cornerstone solution to track, classify, and manage them.

7. REFERENCES

[1] Choi, S.-M., Ko, S.-K., & Han, Y.-S. (2012). A movie recommendation algorithm based on genre correlations. *Expert Systems with Applications*, *39*(9), 8079–8085. https://doi.org/10.1016/j.eswa.2012.01.132

[2] Covington, P., Adams, J., & Sargin, E. (2016). Deep Neural Networks for YouTube Recommendations. *Proceedings of the 10th ACM Conference on Recommender Systems*, 191–198. https://doi.org/10.1145/2959100.2959190

[3] Culley, T. M. (2017). The frontier of data discoverability: Why we need to share our data1. *Applications in Plant Sciences*, *5*(10), 1700111. https://doi.org/10.3732/apps.1700111

[4] Das, S. (n.d.). *DataHub: Popular metadata architectures explained*. Retrieved August 30, 2023, from https://engineering.linkedin.com/blog/2020/datahub-popular-metadata-architectures-explained

[5] *EUDML | Representation of a finite graph by a set of intervals on the real line*. (n.d.). Retrieved August 30, 2023, from https://eudml.org/doc/213681

[6] Heinis, T., & Alonso, G. (2008). Efficient lineage tracking for scientific workflows. *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 1007–1018. https://doi.org/10.1145/1376616.1376716

[7] Holzinger, M. J., Chow, C. C., & Garretson, P. (n.d.). *A Primer on Cislunar Space*.

[8] Ikeda, R., & Widom, J. (n.d.). *Data Lineage: A Survey*.

[9] Isinkaye, F. O., Folajimi, Y. O., & Ojokoh, B. A. (2015). Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, *16*(3), 261–273. https://doi.org/10.1016/j.eij.2015.06.005

[10] Koutras, C., Hai, R., Psarakis, K., Fragkoulis, M., & Katsifodimos, A. (2022). *SiMa: Effective and Efficient Data Silo Federation Using Graph Neural Networks* (arXiv:2206.12733). arXiv. https://doi.org/10.48550/arXiv.2206.12733

[11] Kraker, P., Schramm, M., & Kittel, C. (2021). Discoverability in (a) Crisis. *ABI Technik*, *41*(1), 3–12. https://doi.org/10.1515/abitech-2021-0003

[12] Magazine, S., & Mola, R. (n.d.). *How Things Work: Space Fence*. Smithsonian Magazine. Retrieved August 30, 2023, from https://www.smithsonianmag.com/air-space-magazine/how-things-work-space-fence-180957776/

[13] Mathis, C. (2017). Data Lakes. *Datenbank-Spektrum*, *17*(3), 289–293. https://doi.org/10.1007/s13222-017-0272-7

[14] *Search OSOidx*. (n.d.). Retrieved August 30, 2023, from https://www.unoosa.org/oosa/osoindex/search-ng.jspx?lf_id=

[15] *Space-Track.org*. (n.d.). Retrieved August 30, 2023, from https://www.space-track.org/

[16] *Unified Data Library* (n.d.). Retrieved August 30, 2023, from https://unifieddatalibrary.com/

[17] Ustun, T. S., Hussain, S. M. S., Kirchhoff, H., Ghaddar, B., Strunz, K., & Lestas, I. (2019). Data Standardization for Smart Infrastructure in First-Access Electricity Systems. *Proceedings of the IEEE*, *107*(9), 1790–1802. https://doi.org/10.1109/JPROC.2019.2929621

[18] Vallado, D. A., & Cefola, P. J. (n.d.). *IAC-12-A6.6.11 TWO-LINE ELEMENT SETS – PRACTICE AND USE*.

[19] Vallado, D., & Cefola, P. (2012). Two-line element sets—Practice and use. *Proceedings of the International Astronautical Congress, IAC*, *7*, 5812–5825.