NSDF-Catalog: Lightweight Indexing Service for Democratizing Data Delivery

Jakob Luettgau
University of Tennessee
Knoxville, TN, USA
jluettga@utk.edu

Giorgio Scorzelli and Valerio Pascucci
University of Utah
Salt Lake City, UT, USA
valerio.pascucci@utah.edu

Glenn Tarcea
University of Michigan
Ann Arbor, MI, USA
gtarcea@umich.edu

Christine R. Kirkpatrick
UC San Diego
La Jolla, CA, USA
christine@sdsc.edu

Michela Taufer
University of Tennessee
Knoxville, TN, USA
taufer@utk.edu

2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC) | 978-1-6654-6087-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/UCC56403.2022.00011

***Abstract*—Across domains massive amounts of scientific data are generated. Because of the large volume of information, data discoverability is a challenge, especially for scientists who have not generated the data or are from other domains. As part of the NSF-funded National Science Data Fabric (NSDF) initiative, we developed a testbed to demonstrate that these boundaries to data discoverability can be overcome. In support of this effort, we identify the need for indexing large-amounts of scientific data across scientific domains. We propose NSDF-Catalog, a lightweight indexing service with minimal metadata that complements existing domain-specific and rich-metadata collections. NSDF-Catalog is designed to facilitate multiple related objectives within a flexible microservice to: (i) coordinate data movements and replication of data from origin repositories within the NSDF federation; (ii) build an inventory of existing scientific data to inform the design of next-generation cyberinfrastructure; and (iii) provide a suite of tools for discovery of datasets for cross-disciplinary research. Our service indexes scientific data at a fine-granularity at the file or object level to inform data distribution strategies and to improve the experience for users from the consumer perspective, with the goal of allowing end-to-end dataflow optimizations.**

***Index Terms*—national science data fabric, scientific data, cloud, high performance computing**

I. Introduction

The amount of scientific data is exploding rapidly as computational resources, research instruments, remote sensing, and environmental sensors grow in fidelity and count. In 2016, Globus [1], [2], for example, reportedly managed 150 PB of data spread across 25 billion files [2]. In 2020 [3], Globus reports showed movement of 100 billion files totaling more than 790 PB of data. Globus live statistics today show more than 180 billion files moved with a transfer volume of 1.7 EB.

While most data is collected in the context of a specific research effort for a group of experts in a given domain, many datasets later turn out to be useful beyond their original purpose. For this reason, countless scientific domains and their sub-fields have set up research data archives [4]–[7]. Because these data archives are often not public, some institutions and operators of research data archives have built domain-specific data repositories to help make their data accessible to other researchers [7], [8]. These efforts are encouraged and sometimes required by funding-agencies [9]–[11]. Furthermore, most publishers promote sharing of research artifacts under the banner of the FAIR principles [12]. Many publishers, for example, include guidance in their editorial policies [13] that increasingly state what relevant research artifacts should be submitted to public data repositories. The general recommendation is to submit data to a domain-specific, community-recognized repository when possible, and only turn to general repositories, if specific ones do not exist. This is a dramatic improvement to discoverability and reuse of scientific data, but it also leads to data fragmentation, ultimately resulting in the scientist's lack of awareness on which repository is the most appropriate to upload generated data or where to search for specific datasets.

The overall trend creates a burden for scientists searching for data to answer scientific queries. Even though several domain-agnostic data aggregators that collect data from one or more sources have been developed [14], [15], the definition of appropriate metadata standards across repositories remains a major challenge, adding further complexity to the indexing and search processes. Generally speaking, enforcing metadata adherence early in the dataset generation is a good strategy because scanning an entire dataset repository to extract metadata can be costly. Considering that Globus reported moving 180 billion files, but most of the domain-specific and even multi-domain repositories are listing only several million files each, it becomes clear that there is a large gap between searchable data and available data.

We propose to fill this gap through NSDF-Catalog, a lightweight, comprehensive indexing data service that complements the above-mentioned existing efforts and their limits. In our design and implementation approach, we follow a paradigm in which we build an inventory of available scientific data. In other words, NSDF-Catalog accelerates the process of

978-1-6654-6087-3/22/$31.00 ©2022 IEEE
DOI 10.1109/UCC56403.2022.00011

Authorized licensed use limited to: The University of Utah. Downloaded on August 02,2023 at 01:10:32 UTC from IEEE Xplore. Restrictions apply.

building inventories of scientific data across multiple domain-specific repositories, while informing the design of cross-cutting cyberinfrastructure.

NSDF-Catalog is an effort within the NSF-funded National Science Data Fabric (NSDF) initiative, a project that gathers scientists, computer scientists, and engineers who share the mission of building a platform-agnostic testbed for democratizing data delivery. This vision requires an inventory service across multiple, independently-collected datasets. On the infrastructure side, the coordination of data storage, transfer, and caching requires a federated inventory similar to the Domain Name System (DNS). On the applications and workflow side, we met with multiple scientists who reported use cases that can benefit from even simple keyword search against filenames, but the effort to develop adapters to various relevant domain-specific repositories in many cases is still prohibitive for them. Similarly, scientists from different domains but also young researchers and students often do not know what domain-specific repositories are most suitable for their scientific queries. The main contributions of our work are as follows:

- We analyze data collections across multiple domain-specific repositories in terms of their number and size of collected data and identify five recurrent patterns that are useful to optimize various search and transfer scenarios.
- We use the empirical analysis to inform the requirements for the design of the NSDF-Catalog architecture.
- We design the NSDF-Catalog microservice architecture for a data catalog that meets the requirements.
- We implement the NSDF-Catalog microservice architecture that is composed of a front-end (Web interface), a back-end (REST API), and a python client library.
- We test our catalog by indexing across eight different repositories demonstrating both the *interoperability* and *scalability* of our NSDF-Catalog for datasets totaling 1.6 billion entries.

The remainder of the paper is structured as follows: In Section II, we analyze various scientific data repositories to define the requirements for our lightweight indexing service. In Section III, we leverage the requirements to develop the NSDF-Catalog microservice architecture. In Section IV, we perform an evaluation of our NSDF-Catalog architecture with respect to scalability and interoperability. In Section V, we discuss related work. Last, we summarize our results and conclude with future work in Section VI.

II. Catalog Requirements

When designing an indexing data service that allows for searching across large amounts of raw scientific research data in multiple repositories, an open question is what requirements such a service should meet. We establish a set of requirements from observations of existing data services and properties in well-known repositories.

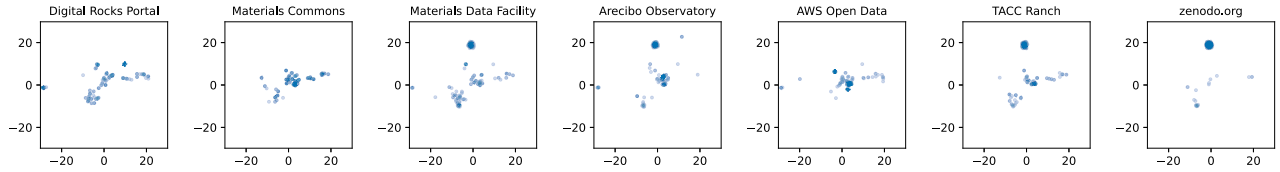
A. Lessons from Established Indexing Tools

We study the performance of established private and public tools providing a wide range of data services such as the Google Data Commons, Dataverse, and Globus to reason about requirements in terms of scale metrics and types of entries (i.e., files or objects) for our indexing catalog. For instance, Google represents data as graphs and uses triples for measuring scale. Recent reports indicate that Google has built two large-scale knowledge graphs one with 850 billion triples containing 200,000 variables and a second with 1.4 trillion triples containing 100,000 variables [16]. While these are certainly impressive upper-bound values in terms of knowledge that can be indexed by Google, they may not be intuitive for scientists who are looking for the location and number of individual raw data entries. Google is also over-counting the number of entries because a single raw data entry may generate many triples. Dataverse reports more intuitive metrics (i.e., it refers to 200,000 collections in 12,000 Dataverses for a total of 2.1 million files). Dataverse is under-counting them because only pre-processed, high-quality data is made available to users who, in some cases, may need to search across all available raw data. Globus on its website reports live statistics indicating that over 180 billion files and 1.7 EB total bytes were transferred. Because of its nature (i.e., indexing movement of entries and not number of individual ones) it can both over-count and under-count entries. Files may be transferred, and thus counted, multiple times. On the other hand, not all files on a storage system may be transferred by Globus and therefore the total files is under-counted. These observations highlight how an indexing catalog should index raw data, and count data entries once no matter whether the data is moved during its lifespan.

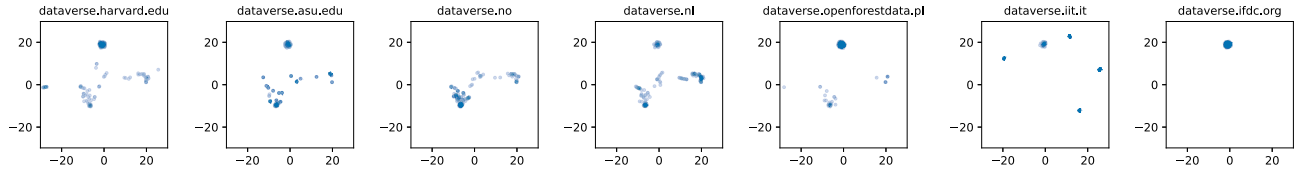
B. Lessons from Properties in Existing Datasets

We study the properties (i.e., number and size of files or objects) and their distributions (i.e., patterns) of existing private and public data repositories to reason about how to best search through collections with our indexing catalog. To this end, we analyze the frequency distributions of 1.6 billion entries from eight repositories, totaling 71.4 PiB of data. The repositories range from domain-specific collections in materials science and astronomy (i.e., Digital Rocks Portal [17], Materials Commons [18], Materials Data Facility [19], and Arecibo [7]) to general data collections such as AWS Open Data [20], TACCs Ranch Long-Term Archive [6], and Zenodo [21] and the Dataverse [22] federation of repositories. Table I describes the repositories in terms of their number of collections, entries, and total entry size. For each collection, we gather an entries' name, creation time, and size. For each repository, we generate a repository fingerprint capturing the relationships between the frequency of entries with a specific size across collections. In other words, each fingerprint serves as a proxy for the fragmentation of data that has to be indexed across collections in the associated repository. For each repository, we create its fingerprint by (i) sampling a subset of collections from a repository, (ii) building a histogram of the distributions and frequency of entry sizes for each sample, and (iii) clustering the histogram. Specifically, the fingerprints are generated with the Uniform Manifold Approximation and

Authorized licensed use limited to: The University of Utah. Downloaded on August 02,2023 at 01:10:32 UTC from IEEE Xplore. Restrictions apply.



(a) Set of domain-specific and general data repositories.



(b) Selected Dataverse repositories.

Fig. 1: Uniform Manifold Approximation and Projection (or UMAP) clustering for different data collections (each represented by a dot) across different, well-known repositories (figures). The clustering is performed using the distribution of number of entries (i.e., files or objects) and their frequency (i.e., histogram of file sizes or object sizes). The plots are a proxy for the repositories' diversity: the more disperse the dots, the more diverse the entries.

TABLE I: Repositories considered in our study grouped in domain-specific for materials science and astronomy, general repositories, and federation of repositories.

Repository	# Collections	# Entries	Size (Bytes)
Digital Rocks Portal	148	17,285	6.1 TiB
Materials Commons	70	258,576	10.2 TiB
Materials Data Facility	178	1,075,706	4.8 TiB
Arecibo Observatory	221	2,045,049	447.4 TiB
AWS Open Data	397	1,617,966,022	50,400.0 TiB
TACC/Ranch	184	1,091,321	20,500.0 TiB
zenodo.org	1,001,459	3,461,517	339.5 TiB
Dataverse	154,472	2,306,495	104.9 TiB

Projection (or UMAP) clustering [23]. Figure 1a presents the fingerprints for the seven repositories (i.e., Digital Rocks Portal, Materials Commons, Materials Data Facility, Arecibo Observatory, AWS Open Data, and TACC Ranch); Figure 1b shows the fingerprints for seven selected repositories from the Dataverse federation. The x and y coordinates in each fingerprint are normalized coordinates of dots representing the repository's diversity in number of entries with a given size. The more dispersed the cluster of dots, the more diverse the collections in the cluster; the darker the cluster, the larger the amount of similar collections.

We observe two different patterns. The first pattern (Pattern 1) consists of a dark dot in the central upper part of the fingerprint – the single cluster indicates a high frequency of identical-in-size entries, with the darker the cluster, the higher the frequency. The second pattern (Pattern 2) consists of a point cloud (i.e., collection of dots) randomly distributed in a centrally located, horizontal stripe – the many clusters indicate the presence of different types of entries. Some clusters in the cloud are darker than others: the darkness represents the frequency of entries of the given size. We can pinpoint these patterns in individual repositories. For instance, in Figure 1a, the three domain-specific repositories for materials sciences share similar entry properties. All three feature mostly collections with many files with different sizes (Pattern 2); the Materials Data Facility also features a large pool of collections with only a single small file in the range of few MB (Pattern 1). Similarly, to Materials Data Facility, Zenodo exhibits Pattern 1 as it is used extensively to publish snapshots of research software composed of many usually small files as well as to publish project reports that often consist only of a single small sized PDF. Contrary to the materials science repositories, Zenodo does not gather a rich set of different-in-size files typical of Pattern 2. Within the Dataverse federation of repositories, we plot the fingerprints of seven selected repositories: four of the largest (i.e., dataverse.harvard.edu, dataverse.asu.edu, dataverse.no, and dataverse.nl), two featuring a small number of clusters (i.e., dataverse.openforestdata.pl and dataverse.lit.ut), and one featuring only a single cluster (i.e., dataverse.lfdc.org).The two largest Dataverse repositories are operated by universities which allow both data and publications to be archived. This explains the presence of both patterns. The primary mission of dataverse.no is to preserve national research data, and hence does not include single file collections, exhibiting only Pattern 2. On the opposite side of the spectrum, dataverse.ifdc.org features only reports, and thus presents only Pattern 1. This observation highlights how our catalog should ultimately preserve effective indexing across the different patterns characterizing the targeted datasets.

C. Summary of Requirements

The resulting requirements that are driving our catalog's design and implementation are as follows.

- Our catalog should index a broad range of individual data entries (i.e., files or objects) ranging from several hundred to millions of data entries.

Authorized licensed use limited to: The University of Utah. Downloaded on August 02,2023 at 01:10:32 UTC from IEEE Xplore. Restrictions apply.

- Our catalog should index raw data and complement existing curated efforts such as Dataverse and domain-specific repositories.
- Our catalog should establish a criterion to identify unique data.
- Our catalog should preserve properties including data such as size, name, collection, and repository.
- Our catalog should preserve effective indexing across the different distribution patterns characterizing the targeted datasets.

III. Catalog Design and Implementation

We design the microservice architecture of our NSDF-Catalog based on identified requirements, best practices, and targeted cyberinfrastructure.

A. Implementation Strategies

We build our architecture using the set of empirical requirements from analysing related services and from sampling the landscape of well-known scientific repositories in Section II. Directly informed by these requirements, we follow these implementation approaches:

- We opt for a minimal and thus *lightweight metadata schema*.
- We aim for a *database agnostic* back-end for which both central and replicated deployments are supported.
- We *decouple harvesting workers and database ingest* to allow asynchronous index updates with many parallel workers.

We further choose to implement the NSDF-Catalog using the following best practices using cloud native technologies to reduce the maintenance burden while increasing the robustness of the catalog's services:

- All sub-services are *containerized*, allowing for both co-located and distributed deployments based on index scales and query loads.
- Services are designed to be *stateless* to simplify load-balancing while avoiding unintended bottlenecks.
- The *deployment automation* enables leveraging different academic and commercial cloud resources.

Finally, our microservice architecture design integrates with other scientific cyberinfrastructure and NSDF-federated services through:

- *Access control* via SciTokens (JSON Web Tokens) to access protected routes and to allow users searching non-public or embargoed repositories and collections.
- *Protected routes* for authorized users and external services to access data preparation as well as usage statistics and analysis.
- *Open source* software and deployment automation to help institutions that do not have resources on site to make their data findable by setting up their own NSDF-catalog instance.

B. Microservice Architecture Overview

Our indexing service is broken up into multiple microservices. We illustrate the high-level overview of the microservices in Figure 2 and their detailed implementation in Figure 3.

Our indexing service supports both ❶ human users and ❷ programmatic access as forms of interaction with our NSDF-Catalog. At the core of our architecture, we place our ❸ REST-API endpoint. Users can deploy either a web front-end or the Python client library to interact with this service, for example, in their Jupyter notebooks. The REST-API endpoint connects to the database endpoint managing the indexing. Our architecture is agnostic to the ❹ database back-end. Our initial prototype and the measurements discussed in Section IV uses Clickhouse as the back-end. Swapping out a database back-end only requires a single file in the REST-API back-end to be changed. We also decouple the harvesting effort from other operations to allow independent scaling and load balancing. As indexing has to be kept up to date continuously, we use an architecture of independent ❺ harvesting workers. Each worker runs in a container and executes an Extract-Transform-Load (ETL) pipeline for a repository. We provide a blueprint for workers that only require added logic to fetch repository entries and transform them into NSDF-Catalog entries. All found entries are then stored into a comma-separated values (CSV) file that are then pushed to a shared storage system (e.g., S3 object storage) and ingested into an index database in a separate process. The ❻ web front-end server can be deployed independently of the REST-API endpoint and both services are stateless.

We provide a default docker-compose configuration that spawns a REST-API endpoint and a front-end server along with a Clickhouse instance. We also include a working configuration for *nginx* to securely serve both the REST-API endpoint and the front-end through a single customizable domain name. The certificates needed to enable HTTPS are obtained using Let's Encrypt/Certbot.

C. Minimal Metadata Schema

Our indexing data service uses the query in Listing 1 as the minimal metadata schema to describe the functional and semantic properties of a repository and allow querying across repositories. The schema includes repository, collection, and name fields needed to uniquely describe a data entry across repositories. The size and *last_modified* fields are additional

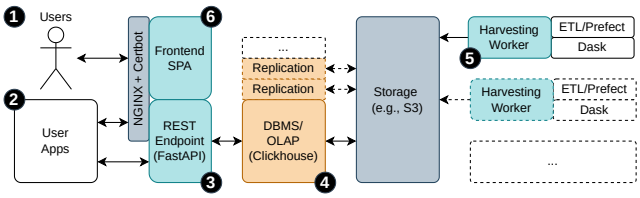


Fig. 2: High-level overview of the NSDF-Catalog microservice architecture, our lightweight indexing service for scientific data.

Authorized licensed use limited to: The University of Utah. Downloaded on August 02,2023 at 01:10:32 UTC from IEEE Xplore. Restrictions apply.

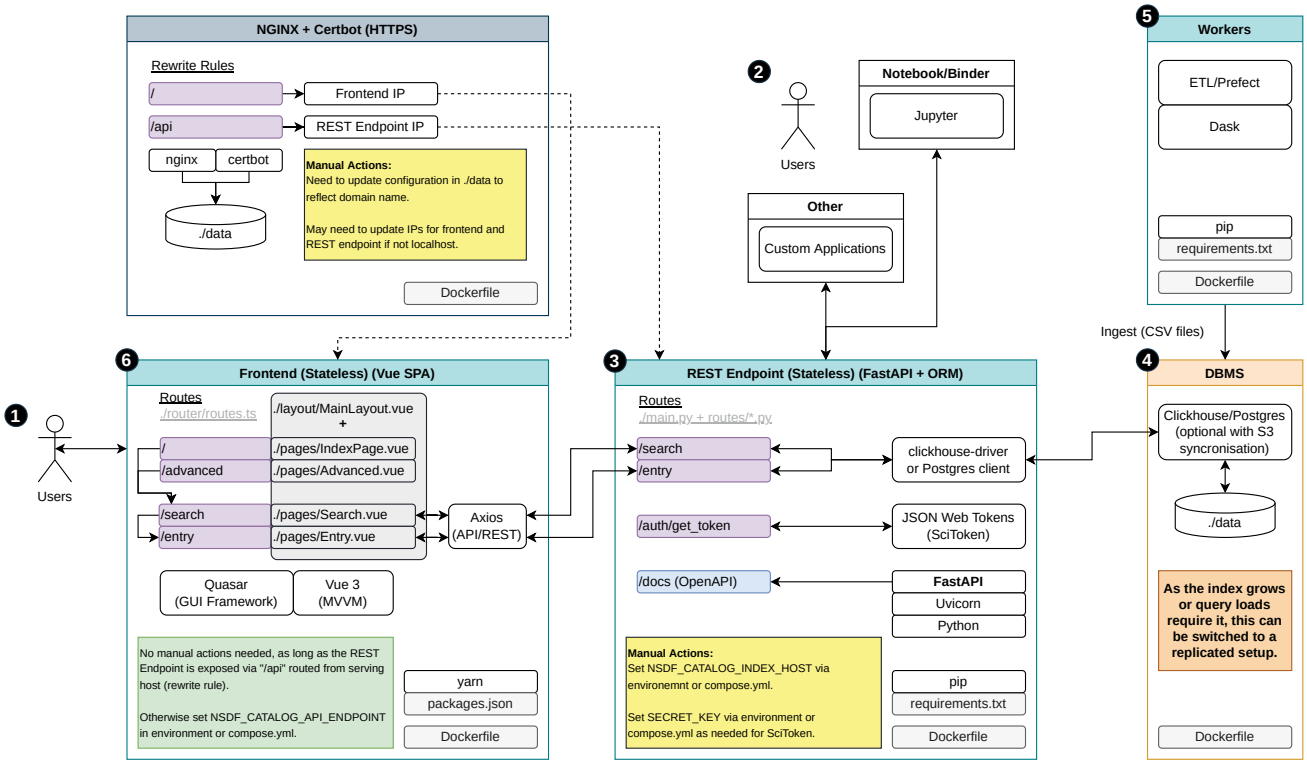


Fig. 3: Whitebox overview of the microservice architecture to support the search indexing and expose it to users and applications.

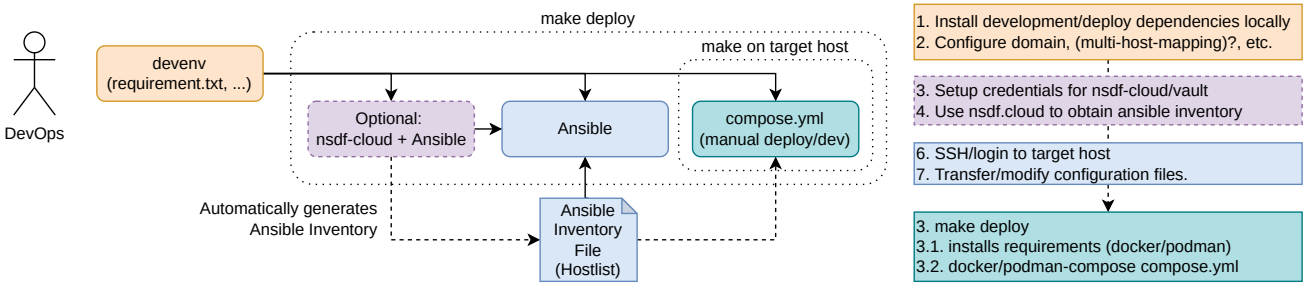


Fig. 4: Deployment automation allowing users to spin up there own index and search with support for academic resources.

metadata needed by other NSDF services to coordinate and optimize data transfers. The *etag* field stores a hash of the data to allow detecting changes in supported repositories.

```
CREATE TABLE nsdf.catalog
(
    `repository` String,
    `collection` String,
    `name` String,
    `size` Int64,
    `last_modified` Nullable(String),
    `etag` Nullable(String)
)
ENGINE = MergeTree
ORDER BY (repository, collection)
SETTINGS index_granularity = 8192
```

Listing 1: Minimal metadata schema used by NSDF-Catalog to index data entries across scientific repositories.

D. Database Back-end

We design the different services to be agnostic about which database back-end can be used. Our prototype uses Clickhouse as the database back-end. Both the front-end and Python client library interact with a catalog instance through the REST-API endpoint. As a result, all logic to interact with the catalog back-end can be consolidated into a single file within the REST-API endpoint source code. As the index grows in size, our catalog can leverage different strategies, including:

- Distributed indexes (e.g., fragmented by repository or scientific domain) to maintain low query latency;
- Replication for load-balancing and scale out as more requests are handled; and
- Hybrid approach that mixes different databases that allows adaptation to more complex schema and queries.

Authorized licensed use limited to: The University of Utah. Downloaded on August 02,2023 at 01:10:32 UTC from IEEE Xplore. Restrictions apply.

E. ETL Workflow for Asynchronous Harvesting and Ingest

For catalog harvesting, we use the well established ETL approach with Prefect [24] and Dask [25] for orchestration. This strategy allows us to decouple harvesting and ingesting into the catalog and makes the service more scalable. To add a new repository we provide a template class that the developer extends with some repository-specific data, to allow for entry enumeration and data extraction to populate our lightweight schema as described in Section III-C. Harvesting workers emit CSV files that are ingested into the global catalog in a separate process.

F. Open Source and Deployment

All NSDF software is open source. The NSDF-Catalog is designed so that it can be deployed in isolation (i.e., without being connected to other NSDF services [26], [27]). This makes our catalog easily deployable using standard tools such as Ansible. Because we include a harvesting worker for NSDF-Catalog instances, connecting to the global NSDF-Catalog is possible at any time.

We illustrate the different deployment modes and their dependencies in Figure 4. Within the NSDF federation we use NSDF-Cloud [28], a unified cloud API to quickly allocate resources for hybrid clouds based on registered credentials. This allows us to support different commercial and academic cloud providers. The NSDF-Cloud utility optionally populates an Ansible inventory file for use by the other deployment stages. Ansible connects to the host and ensures a suitable container runtime is installed. On the host systems our microservices' run inside containers. In the simplest case we use docker-compose. For manual deployment and development purposes we also provide make targets to spawn all our individual services.

IV. Evaluation

We evaluate the NSDF-Catalog by answering the following questions:

- Can we harvest data from different repositories? Answering this question allows us to demonstrate interoperability of our ETL pipeline across different repositories. It also helps us to understand the performance requirements needed to operate the NSDF-Catalog.
- How fast can we ingest data produced by harvesters into an existing index and across different repositories? Answering this question allows us to identify architecture limits due to, for example, the technologies used for the back-end.
- How does the NSDF-Catalog querying performance change against a catalog populated with real-world data? Answering this question allows us to better understand the cost of operation for a service like NSDF-Catalog and also informs load-balancing strategies when distributing the index.

A. Testing Environments

We run our tests using two different cloud providers: Amazon Web Services (AWS) for the harvesting workers and Jetstream2 [29] for benchmarking the NSDF-catalog performance. For the measurement of the harvesting rates, we use the AWS c5d.9xlarge instance type in the us-east-1 region. This instance type features 36 vCPUs with 72.0 GiB of RAM as well as a 900 GB NVMe SSD and a 10 Gigabit network connection. For the performance evaluation, we use the Jetstream2 academic cloud. We use different instance types for Jetstream2 and list the specifications for each of the instances in Table II. All nodes in Jetstream2 are based on AMD Milan 7713 CPUs. Volumes are served by a Ceph cluster with 14 PB of storage.

B. Interoperability Study

To understand the sustained operational performance of our indexing service, we study the harvesting of data sources when searching for new entries and updating existing ones across six of the eight repositories in Table I (i.e., Digital Rocks Portal, Materials Common, Materials Data Facility, AWS Open Data, Zenodo, and Dataverse). We measure the performance of the search and update operations to understand the resource requirements to harvest different repositories. Through this study, we demonstrate that our ETL pipeline is interoperable as it can harvest from a wide range of different repositories.

The NSDF-Catalog's harvesters use Python for repository-specific handling logic. Specifically, we use the client libraries available through the Python Package Index (PyPI). The effort to harvest data can vary significantly depending on the repository and its APIs. For instance, the Digital Rocks Portal does not expose any public API and the harvester has to scrape (i.e., crawl the repository loading and parsing all the entries) the HTML pages using Beautiful Soup [30]. On the other hand, Materials Commons features a Python client library (materials-commons on PyPI) and a REST-API. The Materials Data Facility also offers a Python client library (mdf-forge on PyPI). Furthermore, the AWS Open Data Registry is managed through a public Git-repository hosted on GitHub that contains metadata files. The metadata references S3 buckets that contain the actual data. For the repository, our harvester reads the YAML-based metadata files and then inspects all the objects in the referenced bucket. When harvesting Zenodo, we rely on the zenodopy library that uses the Zenodo REST-API. This

TABLE II: Specifications for instance types on Jetstream2.

instance-type	CPUs	RAM	Root Disk	Ephemeral Disk
m3.tiny	1	3 GB	20 GB	none
m3.small	2	6 GB	20 GB	none
m3.quad	4	15 GB	20 GB	none
m3.medium	8	30 GB	60 GB	none
m3.large	16	60 GB	60 GB	none
m3.xl	32	125 GB	60 GB	none
m3.2xl	64	250 GB	60 GB	none

Authorized licensed use limited to: The University of Utah. Downloaded on August 02,2023 at 01:10:32 UTC from IEEE Xplore. Restrictions apply.

TABLE III: Harvesting performance and network traffic statistics for different repositories.

Repository	Digital Rock Portal	Materials Common	Materials Data Facility	AWS Open Data	Zenodo	Dataverse
# Collections	154	81	637	398	1,012,474	150,834
Collections/s	0.05	1.066	0.799	0.001	51.306	8.236
Upload Bytes/Collection	162.0 KiB	132.0 KiB	320.0 KiB	384.7 MiB	45 Bytes	17.8 KiB
Download Bytes/Collection	725.0 KiB	944.7 KiB	2.0 MiB	1.0 GiB	4.1 KiB	168.2 KiB
# Entries	139,393	131,091	1,206,801	1,540,162,975	3,485,074	2,596,905
Entries/s	45.257	1724.882	1514.179	5232.136	176.603	141.807
Upload Bytes/Entry	183 Bytes	83 Bytes	172 Bytes	104 Bytes	13 Bytes	1.0 KiB
Download Bytes/Entry	820 Bytes	597 Bytes	1.1 KiB	279 Bytes	1.2 KiB	9.8 KiB
Avg. Up	8.1 KiB	140.7 KiB	255.7 KiB	532.7 KiB	2.3 KiB	146.3 KiB
Avg. Down	36.3 KiB	1006.8 KiB	1.6 MiB	1.4 MiB	209.7 KiB	1.4 MiB
Down/Up Ratio	4.476758	7.154878	6.254201	2.681026	91.409164	9.4708
Total Time (seconds)	51	1	13	4,906	328	305

requires a Zenodo access token before any request is handled. Zenodo imposes strict rate limits of 60 requests per minute and 2000 requests per hour that we have to adapt to in the harvesting process. For Dataverse we use the same handler for all 43 tested Dataverse installations in combination with their restful search API.

The complexity of the harvesting process and the structure of the data in a repository have an impact on the harvesting rate of the repository. In Table III we report the harvesting rates (collections per second and entries per second) together with the average upload/download traffic per collection and entry respectively as well as total harvesting time in minutes. All measurements are performed using a single worker. When harvesting the Digital Rocks Portal, we observe an average upload traffic of 8.1 KiB/s and download traffic of 36.3 KiB/s. The harvesting rate is significantly lower and the accumulated network traffic per collection and entry is no higher than with the other repositories. This maybe be either the result of network or rate limits at the repository server or the larger effort it takes to parse and extract information from the HTML representation. In Section II-B we notice how the Materials Commons and the Materials Data Facility repositories share similar fingerprint patterns. We also observe the similarity in the harvesting performance: both repositories perform at similar harvesting rates (one collection per second and about 1,600 entries per second). For Materials Commons we observe average upload traffic of 140.7 KiB/s and download traffic of 1.1 MiB/s; for the Materials Data Facility we observe average upload traffic of 255.7 KiB/s and download traffic of 1.6 MiB/s. The fact that the average upload/download rates are similar suggests to us that the observed behavior can be explained by the similar structures of the two repositories.

For the general data repository we see two repository on opposite ends of the performance spectrum. On one hand, for AWS Open Data we observe a low harvesting rate of only 0.001 collections per second but a very high harvesting rate of 5,200 entries per second. A closer inspection in line with our observations discussed in Section II-B outlines how AWS Open Data counts fewer collections but each collection features 3.9 million entries on average. For AWS Open data we observe an average upload rate of 532.7 KiB/s and download rate of at 1.4 MiB/s. Zenodo exhibits an average of 3 entries per collection and as a result the higher harvesting rate of 51.3 collections per second. Zenodo allows to request batches of up to 10,000 collections or entries; this results in a favorable upload to download ratio. We observe an average upload rate of 2.3 KiB/s and an download rate of 209.7 KiB/s.

Finally, for Dataverse, which has both general and specialized domain-specific repositories, we observe the second highest collection harvesting rate of 8.2 collections per second and 141.8 entries per second. Overall Dataverse also tends to feature a lower average of 17 entries per collection but this depends on the particular Dataverse repository. We observe an average upload rate of 146.3 KiB/s and an download rate of 1.4 MiB/s.

Based on the performance collected while harvesting data across the different repositories, we confirm the benefits of our approach to decouple the harvesting process from other operations. The decoupling allows us to adapt to the different harvesting rates observed for the different repositories. The analysis also gives us a better understanding of the system requirements for different harvesters. For instance, a user does not need all the processing power of the AWS c5d.9xlarge instance type for the harvesting process as many repositories do not saturate the available network connection.

C. Database Back-end Performance Study

The NSDF-Catalog architecture can combine different technologies to ensure scalability and responsiveness of services as the index of scientific data grows. For our prototype we use Clickhouse, a columnar datastore, as our database back-end because it features high-ingestion rates and is tailored to enable aggregation across large amounts of data. We measure the performance to ingest data as the index grows and break down our study into the following two aspects. First, to understand the performance behavior of the back-end under resource constraints, we fill up a m3.tiny (20 GiB) Jetstream2 instance and measure the ingestion performance, index size, and disk fill level under this stress condition. Second, to understand the performance behavior of ingestors for different instances and

Authorized licensed use limited to: The University of Utah. Downloaded on August 02,2023 at 01:10:32 UTC from IEEE Xplore. Restrictions apply.

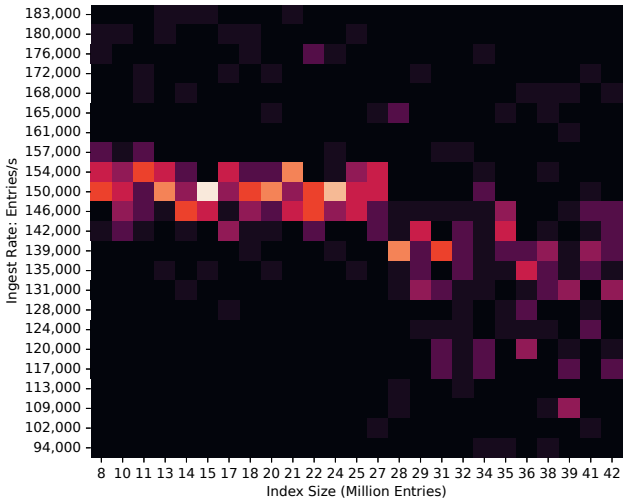
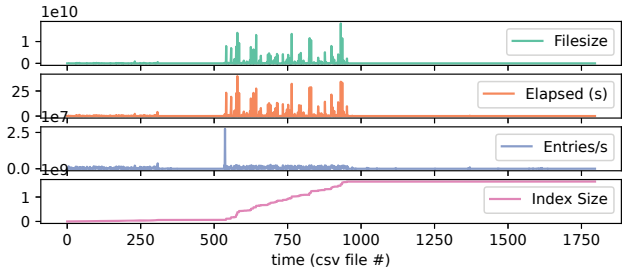
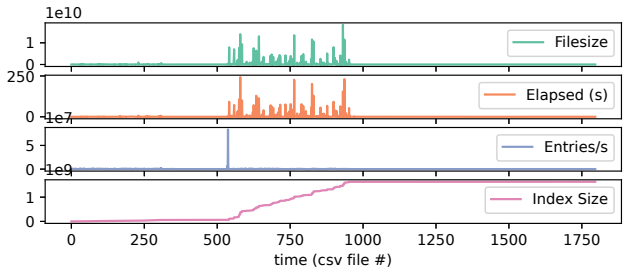


Fig. 5: Performance degradation for ingest rates as the index grows on an m3.tiny instance.



(a) m3.tiny



(b) m3.xl2

Fig. 6: Profile of the workload over time ingesting 1.6 billion entries from 1797 harvested CSV files.

their available memory, we use a 4 TB volume and ingest 1.6 billion entries (the cumulative entries of all the repositories in Table I) in each of the instances in Table II.

a) Understanding Performance Behavior under Resource Constraints: To understand the performance and limiting factors when using Clickhouse as our database back-end we use the smallest instance type m3.tiny that Jetstream2 offers and fill to 95% of its 20 GiB disk volume. As new entries are added to the index, we track the following metrics: ingestion performance, index size, and disk fill level. We plot our results in Figure 5. On the x-axis we show the index size and on the y-axis we plot the ingestion performance (entries per second). Initially, the ingestion performance is stable with occasional outliers at about 150,000 entries added per second. As we approach an index of 28 million entries, we observe average performance drops to a lower level with higher variability to the downside. After closer investigation, this is likely due to the use of the MergeTree Table engine in Clickhouse. By comparing the disk filling level around 28 million and 42 million marks, it appears that Clickhouse triggers housekeeping routines including releases of storage as the MergeTrees are restructured and data is compressed. From this analysis, we learn that by using the smallest instance type we can achieve ingest rates of about 150,000 entries per second and that performance degrades as we approach the system limits because of Clickhouse's housekeeping routines.

b) Understanding Ingest Performance Behavior with different Instance Types: To understand how the NSDF-Catalog's performance scales as we add new resources, we first eliminate the constraint on storage capacity by attaching a 4 TB volume. The volume offers high performance as it is backed by a 14 PB Ceph cluster. Then we measure the ingest performance of all 1.6 billion entries (the cumulative entries of all the repositories in Table I). We use the `clickhouse-client` application within the Clickhouse container and record the time before and after the Clickhouse's command-line CSV import in Listing 2 is executed.

```
query="INSERT INTO nsdf.catalog FORMAT CSV"
cat ${filename} | clickhouse-client --query=${query}
    " --time ... <clickhouse credentials>
```

Listing 2: Clickhouse's command-line CSV import.

The commands read the CSV file and pipes its content to the clickhouse client that then imports the data into the index. Each CSV represents the output of one of the harvesting workers which typically generates one CSV file per repository. Two exceptions to this approach are: AWS Open Data, where each S3 bucket results in a separate CSV file, and Dataverse, where each Dataverse installation results in its own CSV file.

We collect performance values for the multiple imports of CSV files in Listing 2 and analyse the ingest performance

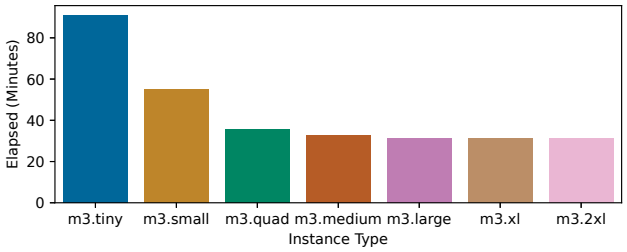


Fig. 7: Total time taken to ingest 1.6 billion entries on different instance types using Clickhouse and Jetstream2.

Authorized licensed use limited to: The University of Utah. Downloaded on August 02,2023 at 01:10:32 UTC from IEEE Xplore. Restrictions apply.

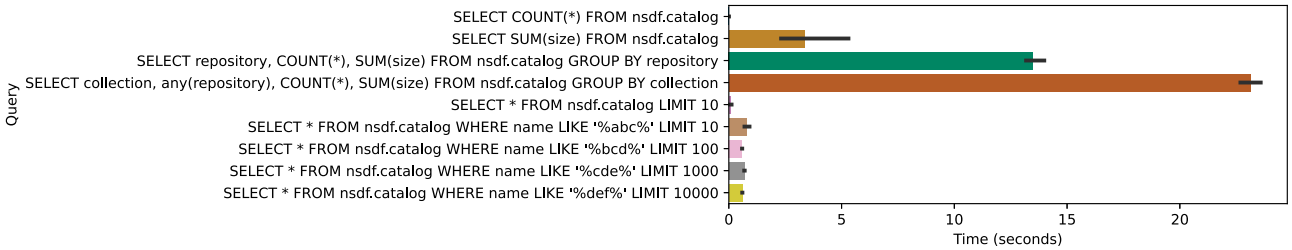


Fig. 8: Query performance against the index populated with 1.6 billion entries using the m3.quad instance type.

for differently sized dumps. In Figure 6 we plot the file size, elapsed time in second, and entries per second from a workload ingesting 1.6 billion entries from 1797 harvested CSV files. Across instances we observe similar permanence behaviors and thus we report here only results for the smallest m3.tiny instance and largest m3.xl2 instance. The performance gained when moving from the smallest to the largest instance is nearly a factor of three. This shows that Clickhouse, without additional tuning, can parallelize the import process. In Figure 7 we report the total time taken to import all files. The figure shows that without further tuning, it is sufficient to run the import using the m3.large instance type with 16 CPUs and 60 GB of RAM. For larger instances we observe diminishing returns: the performance is almost identical to the m3.large instance type. We also learn that performance gains are tangible only when importing sufficiently large CSV files for which Clickhouse performs some parallelization when importing the large files. For our architecture, this means that harvesting workers should be configured to generate large CSV files to take advantage of this performance gain.

D. Query Performance Study

Querying data across large tables can be an expensive operation. We build NSDF-Catalog to support a wide range of queries meeting the following three scenarios.

First, NSDF-Catalog should support other NSDF services that coordinate data movements and replication of data from origin repositories within the federation. Queries related to this activity are composed of simple ”WHERE” clauses. For a particular collection, queries can explore individual entries or all entries. Queries can request to search the full repository, a collection, or an entry name and its size.

Second, to better understand the inventory of existing scientific data and improve system designs, NSDF-Catalog has to generate statistics. Queries related to this activity should perform ”GROUP BY” operations along with various aggregations such as counting and building sums.

Finally, as a tool for discovery of available data for users worldwide, the catalog should serve researchers asking for all matches through large repositories. Thus, queries related to this activity should support complex ”WHERE” clauses.

We define nine queries that cater to these three usage scenarios and repeatedly run the query against the index populated with all 1.6 billion entries on an m3.quad instance. Similar to our previous analysis, m3.tiny performance is lower, but the difference for the other instance types is small. We take 10 measurements for each query and report the average execution time in seconds. We clean caches between measurements.

The results for the different queries are shown in Figure 8. Queries counting all entries return on average within 10 ms; the query taking the SUM across all entries requires 6 seconds on average. For GROUP BY operations, we observe the highest query latency, requiring 14 seconds to generate aggregate statistics for all repositories and 24 seconds for aggregate statistics on all collections. All queries with aggregations show higher variability. For searches against the index asking for the first 10 entries returns within 244 ms; a specific search independent of the LIMIT return within about 1 second. These results demonstrate that query performance even with the specification of the m3.quad featuring only 4 CPUs and 15 GB of RAM provides reasonable querying performance.

V. Related Work

Data repositories and data catalogs have been used across scientific domains for many years. Many of these repositories are domain-specific such as [7], [18], [31]; a large number of efforts provide cross-disciplinary solutions. For instance, Dataverse [22] and Rucio [31] are open-source efforts initially funded by institutions in need of a solution otherwise unfulfilled and currently are used by the scientific communities around the globe. Other initiatives typically establishing meta catalogs are enjoying national or government backing, for example, data.gov [32] and EUDAT [15]. Besides community and national efforts there are also corporate catalogs or data replication efforts such as Google’s Data Commons [16] and AWS Open Data Repository [20]. Our effort complements all these systems to create an inventory of available scientific data. The NSDF-catalog uses a system of systems approach that can support organizations to make their raw data searchable, even before investing into the effort of getting included in more curated catalogs. In our discussion on lessons from established indexing tools in Section II-A we cover key features of indexing tools used in Google Data Commons or Dataverse, or data transfer tools such as Globus, outline how our NSDF-Catalog builds upon these tools and complements them in their functionalities.

Authorized licensed use limited to: The University of Utah. Downloaded on August 02,2023 at 01:10:32 UTC from IEEE Xplore. Restrictions apply.

VI. Summary & Conclusion

We present the NSDF-Catalog, a lightweight indexing service for searching row data across repositories. We design the catalog based on lessons learned from established indexing tools and properties in existing datasets. Our implementation builds on a decoupled microservice architecture. We demonstrate the catalog's performance indexing up to 1.6 billion entries referencing about 71.8 PiB of data across eight repositories. Our results show that NSDF-Catalog can offer a lightweight service to make cross-disciplinary data searchable efficiently.

In future work we will extend our index with a larger suite of statistics to identify scientific content in data. In particular, we will leverage scientific content to prefetch and stage data across the NSDF federation. We also observe how a significant number of files are self-describing data formats or archives both of which are composed of sub-structures. Leveraging the sub-structures to improve the search performance is another future direction.

Acknowledgment

This research was supported by the National Science Foundation (NSF) under grant numbers #1841758, #2028923, #2103845 and #2138811; the Extreme Science and Engineering Discovery Environment (XSEDE) under allocation TG-CIS210128; Chameleon Cloud under allocation CHI-210923; and IBM through a Shared University Research Award.

Research Artifacts

Analysis code and corresponding datasets are available for download at: https://doi.org/10.5281/zenodo.7268442

References

[1] I. Foster, "Globus Online: Accelerating and Democratizing Science through Cloud-Based Services," *IEEE Internet Computing*, vol. 15, no. 3, pp. 70–73, May 2011.

[2] K. Chard, S. Tuecke, and I. Foster, "Globus: Recent Enhancements and Future Plans," in *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*. Miami USA: ACM, Jul. 2016, pp. 1–8.

[3] "January 2020: Globus Turns 10...18 — globus," https://www.globus.org/blog/january-2020-globus-turns-1018.

[4] NCAR, "NCAR's Research Data Archive," https://rda.ucar.edu/.

[5] R. Petrie, S. Denvil, S. Ames, G. Levavasseur, S. Fiore, C. Allen, F. Antonio, K. Berger, P.-A. Bretonnière, L. Cinquini, E. Dart, P. Dwarakanath, K. Druken, B. Evans, L. Franchistéguy, S. Gardoll, E. Gerbier, M. Greenslade, D. Hassell, A. Iwi, M. Juckes, S. Kindermann, L. Lacinski, M. Mirto, A. B. Nasser, P. Nassisi, E. Nienhouse, S. Nikonov, A. Nuzzo, C. Richards, S. Ridzwan, M. Rixen, K. Serradell, K. Snow, A. Stephens, M. Stockhause, H. Vahlenkamp, and R. Wagner, "Coordinating an operational data distribution network for CMIP6 data," *Geoscientific Model Development*, vol. 14, no. 1, pp. 629–644, Jan. 2021.

[6] TACC, "Ranch - Texas Advanced Computing Center," https://www.tacc.utexas.edu/systems/ranch.

[7] Arecibo Observatory, "Arecibo Observatory Data Catalog," https://www.naic.edu/datacatalog/.

[8] NASA, "NASA Open Data Portal," https://data.nasa.gov/.

[9] M. Kozlov, "NIH issues a seismic mandate: Share data publicly," *Nature*, vol. 602, no. 7898, pp. 558–559, Feb. 2022.

[10] European Commision, "Data Guidelines — Open Research Europe," https://open-research-europe.ec.europa.eu/for-authors/data-guidelines.

[11] D. Forschungsgemeinschaft, "Guidelines for Safeguarding Good Research Practice. Code of Conduct," Apr. 2022.

[12] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. G. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. C. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific Data*, vol. 3, no. 1, p. 160018, Mar. 2016.

[13] "Data Repository Guidance — Scientific Data," https://www.nature.com/sdata/policies/repositories.

[14] "DataCite Repository Selector," https://repositoryfinder.datacite.org/.

[15] EUDAT, "B2FIND - Interdisciplinary discovery portal in the EUDAT Service Catalogue," http://b2find.eudat.eu/.

[16] Data Commons, "Data Commons 2022," https://www.datacommons.org/.

[17] M. Prodanovic, M. Esteva, and M. Hanlon, "Digital Rocks Portal," Sep. 2015.

[18] B. Puchala, G. Tarcea, E. A. Marquis, M. Hedstrom, H. V. Jagadish, and J. E. Allison, "The Materials Commons: A Collaboration Platform and Information Repository for the Global Materials Community," *JOM*, vol. 68, no. 8, pp. 2035–2044, Aug. 2016.

[19] B. Blaiszik, K. Chard, J. Pruyne, R. Ananthakrishnan, S. Tuecke, and I. Foster, "The Materials Data Facility: Data Services to Advance Materials Science Research," *JOM*, vol. 68, no. 8, pp. 2045–2052, Aug. 2016.

[20] AWS, "Open Data on AWS," https://aws.amazon.com/opendata/.

[21] European Organization For Nuclear Research and OpenAIRE, "Zenodo," 2013.

[22] G. King, "An Introduction to the Dataverse Network as an Infrastructure for Data Sharing," *Sociological Methods & Research*, vol. 36, no. 2, pp. 173–199, Nov. 2007.

[23] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction," Sep. 2020.

[24] Prefect Technologies, "Prefect," Prefect, Aug. 2022.

[25] M. Rocklin, "Dask: Parallel computation with blocked algorithms and task scheduling," in *Proceedings of the 14th Python in Science Conference*, no. 130-136. Citeseer, 2015.

[26] NSDF, "National Science Data Fabric," https://nationalsciencedatafabric.org/.

[27] P. Olaya, J. Luettgau, N. Zhou, J. Lofstead, G. Scorzelli, V. Pascucci, and M. Taufer, "NSDF-FUSE: A Testbed for Studying Object Storage via FUSE File Systems," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '22. New York, NY, USA: Association for Computing Machinery, Jun. 2022, pp. 277–278.

[28] J. Luettgau, P. Olaya, N. Zhou, G. Scorzelli, V. Pascucci, and M. Taufer, "NSDF-Cloud: Enabling Ad-Hoc Compute Clusters Across Academic and Commercial Clouds," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '22. New York, NY, USA: Association for Computing Machinery, Jun. 2022, pp. 279–280.

[29] D. Y. Hancock, J. Fischer, J. M. Lowe, W. Snapp-Childs, M. Pierce, S. Marru, J. E. Coulter, M. Vaughn, B. Beck, N. Merchant, E. Skidmore, and G. Jacobs, "Jetstream2: Accelerating cloud computing via Jetstream," in *Practice and Experience in Advanced Research Computing*, ser. PEARC '21. New York, NY, USA: Association for Computing Machinery, Jul. 2021, pp. 1–8.

[30] "Beautiful Soup: We called him Tortoise because he taught us." https://www.crummy.com/software/BeautifulSoup/, 2004.

[31] M. Barisits, T. Beermann, F. Berghaus, B. Bockelman, J. Bogado, D. Cameron, D. Christidis, D. Ciangottini, G. Dimitrov, M. Elsing, V. Garonne, A. di Girolamo, L. Goossens, W. Guan, J. Guenther, T. Javurek, D. Kuhn, M. Lassnig, F. Lopez, N. Magini, A. Molfetas, A. Nairz, F. Ould-Saada, S. Prenner, C. Serfon, G. Stewart, E. Vaandering, P. Vasileva, R. Vigne, and T. Wegner, "Rucio: Scientific Data Management," *Computing and Software for Big Science*, vol. 3, no. 1, p. 11, Dec. 2019.

[32] "Data.gov," https://www.data.gov/.

Authorized licensed use limited to: The University of Utah. Downloaded on August 02,2023 at 01:10:32 UTC from IEEE Xplore. Restrictions apply.