SN Computer Science (2024) 5:462
https://doi.org/10.1007/s42979-024-02789-2

ORIGINAL RESEARCH

The Data Analytics Framework for XDMoD

Aaron Weeden[1] · Joseph P. White[1] · Robert L. DeLeon[1] · Ryan Rathsam[1] · Nikolay A. Simakov[1] · Conner Saeli[1] · Thomas R. Furlani[1]

Received: 1 October 2023 / Accepted: 12 March 2024

Abstract

Open XDMoD is an established, web-based software tool that facilitates monitoring of cyberinfrastructure (CI); including metrics on compute job performance, cloud usage, storage, science gateways, users, institutions, allocations, awards, and more. Centers can install Open XDMoD and populate it with their local CI information. Additionally, information is acquired daily for CI systems allocated by the National Science Foundation (NSF) Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program and placed in an extensive historical data warehouse as part of an ACCESS-hosted version of Open XDMoD. A user portal displays customizable charts that can be filtered and drilled down to provide a wide variety of reports quickly and easily. However, there will always be desired analyses for which the data are available but which cannot be performed because of limitations in the user interface. To remedy this situation, we have developed a Data Analytics Framework for XDMoD that provides an application programming interface (API) to the Open XDMoD data warehouse; enabling CI studies, user studies, return on investment studies, and any other studies that can utilize the data. The Data Analytics Framework provides data scientists the capability to access data from Open XDMoD and perform any desired analyses using analytics tools of their choice. The framework is available both for the ACCESS-hosted version of Open XDMoD for broad-ranging studies of national CI and for centers' installations of Open XDMoD for studies of local CI.

Keywords Open XDMoD · Metrics · Cyberinfrastructure · Jupyter notebooks

Introduction

Science and Engineering (S &E) is a major component of the US budget constituting 3.4% of the US GDP in 2021 [1]. The National Science Foundation (NSF) invests in a national cyberinfrastructure (CI) ecosystem that enables a broad and diverse set of requirements, users, and usage modes from all areas of S &E research and education. One major component of this are the CI resources allocated by the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program [2]. Comprehensive instrumentation, monitoring, measurement, and reporting across all layers of the systems and services making up the ACCESS ecosystem are essential to providing the situational awareness necessary for achieving increased levels of efficiency, understanding, autonomous operations, robustness, and performance. The ACCESS Monitoring & Measurement Services (MMS) team serves in the important role of monitoring the ACCESS CI to ensure optimal performance, robustness, and usage (including compute, cloud, storage,

This article is part of the topical collection "Metrics for Measuring Success of CyberInfrastructure (CI) Projects" guest edited by Ritu Arora and Amit Majumdar.

✉ Aaron Weeden
aaronwee@buffalo.edu

Joseph P. White
jpwhite4@buffalo.edu

Robert L. DeLeon
rldeleon@buffalo.edu

Ryan Rathsam
ryanrath@buffalo.edu

Nikolay A. Simakov
nikolays@buffalo.edu

Conner Saeli
connersa@buffalo.edu

Thomas R. Furlani
furlani@buffalo.edu

[1] Center for Computational Research, University at Buffalo, Buffalo, NY 14203, USA

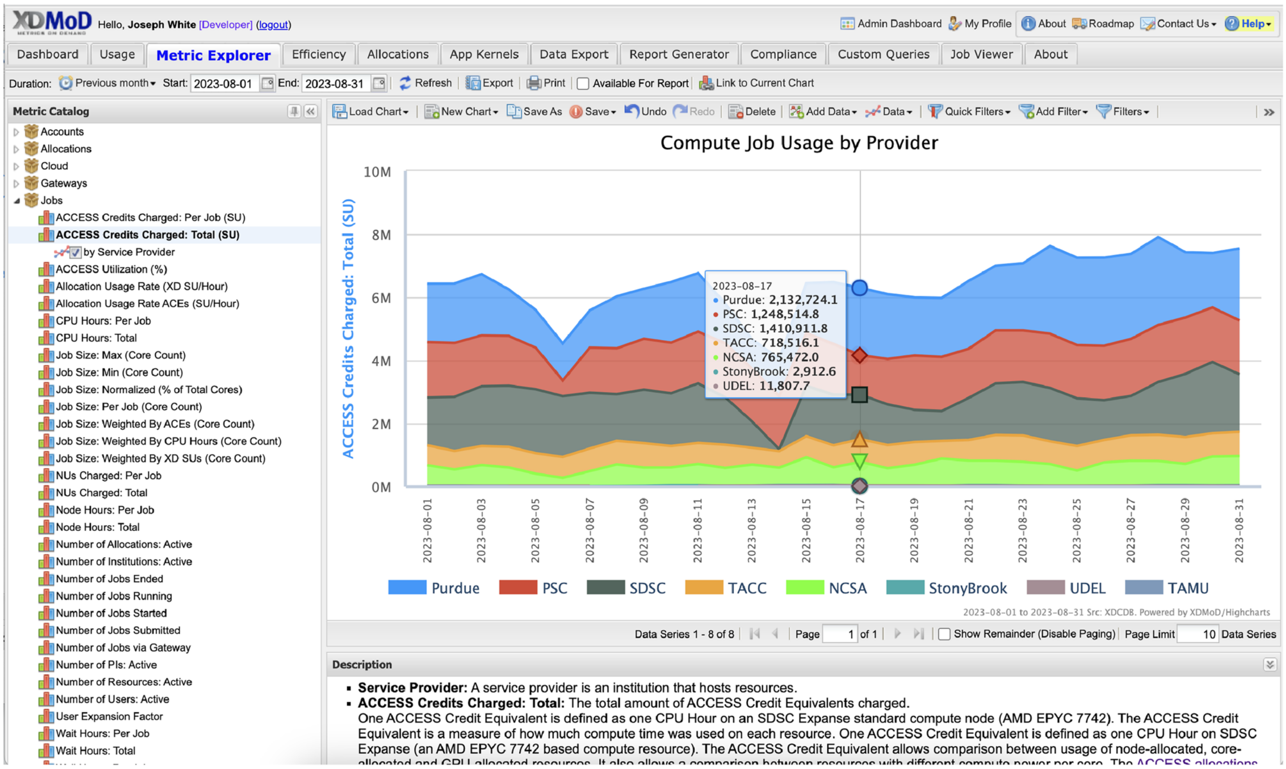
Published online: 20 April 2024

networking, software/data services, etc.). ACCESS MMS also provides services to several other related NSF-funded programs such as Campus Cyberinfrastructure (CC*) [3] and Cyberinfrastructure for Sustained Scientific Innovation (CSSI) [4].

ACCESS MMS is an extension of the successful Technology Auditing Service (TAS) and XD Metrics Service (XMS) programs that monitored the resources allocated by the NSF eXtreme Science and Engineering Discovery Environment (XSEDE) [5] and tracked historical usage data dating back as far as the NSF TeraGrid [6] program. The main utility ACCESS MMS uses for ingesting and reporting data is Open XDMoD [7], which provides a web-based portal with capabilities for data exploration, visualization, and export. Open XDMoD's graphical user interface allows users to make charts rapidly and easily, to export data, and to construct reports on various aspects of CI usage and functionality. Open XDMoD is role-based, allowing the various CI stakeholders access to the data of most use to them. Researchers can see data from their own work, principal investigators (PIs) can see the data pertaining to their group, center directors and support staff can see the data across their center, and program officers can see all of the data.

Figure 1 shows a screenshot of the Open XDMoD portal illustrating some of the user interface features. The interface is tab-based with different data analysis and visualization components available in different tabs. The screenshot shows the *Metric Explorer* tab that allows easy generation of a wide variety of interactive charts. It has detailed, embedded data descriptions and an easy-to-navigate, tree-based component for exploring the various data available in the Open XDMoD data warehouse.

The Open XDMoD data warehouse is a set of databases that employs a dimensional starflake model [8] for storage of CI data. Data are organized into fact tables, dimension tables, and aggregate tables. The data are obtained via data processing pipelines that involve parsing log files (e.g., Slurm scheduler logs, web server logs, performance data archives) or relational databases such as PostgreSQL, normalizing the data, and loading them into fact and dimension tables. Open XDMoD's data warehouse architecture is optimized for fast, interactive data retrieval via the portal. To achieve this, the fact data are aggregated across different time ranges (day, month, quarter and year) into aggregate tables. Data are organized into realms (e.g., *Jobs*, *Cloud*, *Storage*), with each realm having its own fact and aggregate



- **Service Provider:** A service provider is an institution that hosts resources.
- **ACCESS Credits Charged: Total:** The total amount of ACCESS Credit Equivalents charged.
One ACCESS Credit Equivalent is defined as one CPU Hour on an SDSC Expanse standard compute node (AMD EPYC 7742). The ACCESS Credit Equivalent is a measure of how much compute time was used on each resource. One ACCESS Credit Equivalent is defined as one CPU Hour on SDSC Expanse (an AMD EPYC 7742 based compute resource). The ACCESS Credit Equivalent allows comparison between usage of node-allocated, core-allocated and GPU allocated resources. It also allows a comparison between resources with different compute power per core. The ACCESS allocations

Fig. 1 Screenshot of the Open XDMoD portal showing the *Metric Explorer* tab that supports interactive charts. The *Metric Explorer* tab has a tree-based catalog of the available data from the data warehouse; an interactive chart panel that supports a wide range of chart types; a toolbar to configure data selection and filtering; and online, context-specific help text. This screenshot shows the ACCESS-hosted version of Open XDMoD that contains data from ACCESS-allocated resources

tables, and dimension tables are shared across realms or individual to each realm. For example, the *Jobs* realm contains data about compute jobs that are run by batch job schedulers typically used in high-performance computing resources. The fact table for the *Jobs* realm contains data such as compute job submit, start, and finish times; how many nodes, CPU processors, GPUs, and memory were used; and to which queue the job was submitted. Fact tables in the data warehouse include foreign keys to rows in the dimension tables; this allows each row to be filtered or grouped by the dimensions. For example, in the fact tables of the *Jobs* realm, each job is keyed to a person who ran the job, a field of science for the project that submitted the job, a resource on which the job ran, etc. The dimension tables store information about people, organizations, resources, fields of science, etc. For example, the resource dimension tables store the name, location, and specifications of compute resources. The metrics and dimensions available in each realm of Open XDMoD are listed in Appendices A, B, and C.

Motivation

Open XDMoD is a well-established tool for CI management that has been developed and supported for more than a decade. However, it has certain constraints and limitations that must be worked around to support many types of analysis and reporting. In this section we highlight some of the desired types of analysis and the limitations of Open XDMoD; these can be overcome via the development of an application programming interface (API), which we discuss in Sect. 3.

Open XDMoD has been successfully used in multiple workload analyses of CI resources and programs, including NCSA Blue Waters [9] and the NSF Innovative HPC program [10]. It has also been used in other studies, such as analysis of compute node sharing on HPC clusters [11], power consumption in HPC [12], and financial return on investment for IT infrastructure [13]. The workload analyses [9] and [10] involved data visualizations in the Open XDMoD portal, data exported from Open XDMoD, data from sources outside of Open XDMoD, and subsequent analysis with other tools (R [14] and Python with NumPy [15] libraries). Some of the workload analyses relied on data obtained via direct SQL queries into the Open XDMoD data warehouse because the data were not available in the portal. Direct queries to the data warehouse circumvent the Open XDMoD portal access controls, so they should not be used as general purpose mechanisms for data retrieval.

The Open XDMoD architecture supports extensive customization to add new data realms, new dimensions to existing realms, and new visualization capabilities to the portal. For example, the Open XDMoD instance that was used to analyze NCSA Blue Waters had customizations to add dimensions for the compute node type and better matching of job size categories to the distribution of job sizes. As another example, the ACCESS-hosted version of Open XDMoD has many customizations including an extra *Custom Query* tab that provides a limited number of hardcoded queries so that NSF program officers and center management staff can view information not otherwise available, such as funding amounts associated with allocated projects. However, this type of customization cannot be performed by end users of the portal and requires knowledge of the Open XDMoD software and time-consuming database reaggregation steps that must be performed by an administrator.

Open XDMoD has a report generator that allows user-configurable reports to be generated and automatically emailed periodically. The report contents can include any chart that is available from the *Metric Explorer* tab in the portal, and the reports can be generated in PDF or Microsoft Word formats. The reports are primarily chart-based; there is no support for tabular data display (which is supported in the *Usage* tab in the portal). Chart titles, subtitles, and axis labels are all configurable, but there is no support for figure captions, footnotes, or blocks of text within the reports. To add any other non-chart content, reports must be exported in Microsoft Word format and then manually edited.

Open XDMoD is widely utilized in academia, industry, and government with over 400 known installations worldwide. The ACCESS MMS team solicits feedback about Open XDMoD from stakeholders via multiple mechanisms, such as an online roadmap to which anyone can post requests, regular XDMoD Users Group meetings, support tickets, and conversations at international conferences such as Practice and Experience in Advanced Research Computing (PEARC) and SC. Common requests include adding more chart types (e.g., scatter plots and histograms) and adding new data realms so that Open XDMoD can be used to analyze data from other sources.

The ACCESS MMS team is developing a CI simulator that will be used to predict the response of the CI ecosystem to proposed new systems or changes in existing systems, enabling efficient deployment and use. This will comprise multiple mathematical models including machine learning models that use the data in the ACCESS-hosted version of Open XDMoD. This task will require a mechanism to programmatically obtain data from the data warehouse and load them into models written in R and Python.

One other constraint of the Open XDMoD data warehouse is that the dimensions are determined at aggregation time. For example, the *SUPREMM* (job-level performance) realm has a dimension for the overall CPU usage for each compute job. This allows easy creation of histogram plots that show resource utilization binned by the percent CPU usage of the compute jobs. However, the bin sizes are fixed at 10% increments from 0 to 100 and cannot be changed by end users in the portal, so histograms with different bin sizes or density plots cannot be created in Open XDMoD. Instead, the job-level data must be exported using the *Data Export* tab and then processed using other software.

Given our experience using Open XDMoD for data analysis and requirements from the community, we decided that creating a programmatic API to the data in the Open XDMoD data warehouse would allow us to address the issues described above. A programmatic API would allow us to make use of existing, external data analysis and visualization tools rather than trying to duplicate them within the Open XDMoD portal.

Design and Implementation

In this section we discuss our design goals for the Data Analytics Framework, which is a new programmatic API for Open XDMoD, and how we implemented its initial version.

Design Goals

The high-level goals for the design of the Data Analytics Framework were (1) to provide a programmatic API to the data in Open XDMoD to facilitate data analysis that could not otherwise be done in the portal, (2) to further broaden the utility of Open XDMoD for CI analysis, and 3) to contribute to the sustainability of the project.

We did a brief evaluation of the tools that were in common use for data analysis including programming languages such as Python, R [14], Julia [16] and visual tools/environments such as RStudio [17], Jupyter [18], and MATLAB [19]. Our team had the most prior experience with using Python and R for developing machine learning models and data processing for workload analysis, so we opted to support APIs for both Python and R.

To derisk the feasibility of a Python API, we developed a basic proof of concept comprising some Python code within a Jupyter notebook [18]. The Python code made requests to existing HTTP endpoints that support the Open XDMoD portal, retrieved comma separated values (CSV), serialized them into Python data structures, and displayed them in plots embedded in the notebook.

The experience gained from developing the proof-of-concept prototype, combined with the high-level requirements from the project team and features requested from Open XDMoD users, led us to identify the following set of main goals:

- Extend and support a documented, stable, versioned HTTP API to the XDMoD data warehouse.
- Support open-source Python and R libraries that use the HTTP API to provide programmatic access to the data in Open XDMoD.
- Support Jupyter notebooks as a framework for using the Python and R libraries.
- Ensure that training materials, example notebooks, and API documentation are available, supported, and open to community contributions.

Extending the existing HTTP API to Open XDMoD would ensure that the existing role-based access controls in Open XDMoD were honoured and could not be circumvented by API calls. We did not set a responsiveness goal for data access endpoints since they could be used for transfers of large amounts of data, but we did want to monitor the impact of any new data access endpoints and how they could impact the existing portal load times.

We realized that comprehensive documentation and training materials would have to form a core part of the design rather than an afterthought at the end of the development process. We also realized that Jupyter notebooks, which have export capabilities and existing open-source data visualization libraries, could be used to significantly improve Open XDMoD's reporting capabilities. We set a goal of having the charts in the notebooks match the look and feel of the Open XDMoD portal where possible (e.g., match chart dataset colors, font size and face, layout, etc.).

The high-level design schematic of the Data Analytics Framework is shown in Fig. 2. To create the Data Analytics Framework, we developed Python and R code, example Jupyter notebooks, and modifications to the Open XDMoD portal code. During this process, we utilized version control

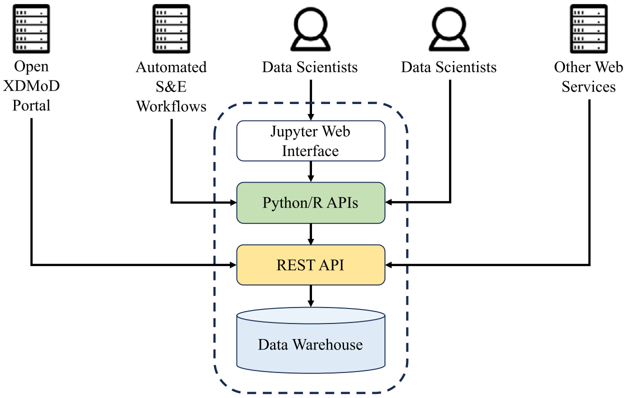


Fig. 2 High level schematic of the Data Analytics Framework for XDMoD. On top of the data warehouse sits an HTTP application programming interface (API). On top of the HTTP API sit Python and R APIs, on top of which sits a Jupyter web interface. Data scientists access the Python and R APIs either directly or through the Jupyter web interface. The Open XDMoD portal and other web services access the HTTP API, and automated science and engineering (S &E) workflows access the Python and R APIs

and automated testing and wrote supporting documentation for end users, system administrators, and developers. We detail these development activities in the following sections.

Python Development

Due to our team having relatively more experience with Python over R, we chose to develop the Python language API first and then begin work on the R version once the initial Python version was complete. In this section we list the details of the Python API. The R API is discussed later in Sect. 5.2.3.

The data that are obtained from the Open XDMoD data warehouse naturally fit into data frames, which are two-dimensional tabular data structures commonly used in data science. The Pandas [20] Python library is a widely used library for data science and provides an API for storing and manipulating data in data frames. It also has methods for exporting data into other formats, which allows users to manipulate the data outside of Pandas or even Python. Thus, we decided that the Python API should provide data in Pandas data frames to support flexibility and extensibility.

To obtain data from the Open XDMoD data warehouse, we decided the Python API would make HTTP requests to endpoints on the web server that runs the Open XDMoD portal. The portal's user interface had already been built around HTTP endpoints; each time a user generates a plot or views a listing of data on the portal, the portal makes HTTP requests to endpoints behind the scenes. The new Python API was designed to utilize the same endpoints plus a few additional ones needed for retrieval of raw fact data. We wrote the code for making HTTP requests on top of the Requests [21] Python library.

We designed the Python API to replicate certain functionality of the Open XDMoD portal. For example, in the *Metric Explorer* tab in the portal, the user is able to obtain specified metrics over a specified duration from specified realms, possibly grouped and/or filtered by specified dimensions, either as timeseries data aggregated by a specified aggregation unit (day, month, quarter, or year) or aggregated across the entire duration. To replicate this functionality in the Python API, we created a method called `get_data()` that has parameters for `metric`, `duration`, `realm`, `dimension`, `filters`, `dataset_type`, and `aggregation_unit`. This method allows the same data to be obtained that are used to make the charts in the *Metric Explorer*. An example of the `get_data()` method is shown below; in this example, data are retrieved for the number of active users each day over a four-month time period who had a compute job finish, grouped by field of science. The resulting data frame contains a column for each field of science and a row for each day, and the cells contain the number of active users in that field of science on that day.

```
data = data_warehouse.get_data(
    duration=('2023-01-01', '2023-04-30'),
    realm='Jobs',
    metric='Number of Users: Active',
    dimension='Field of Science',
    dataset_type='timeseries',
    aggregation_unit='Day'
)
```

The data from the `get_data()` method are retrieved from the aggregate tables of the Open XDMoD data warehouse. The aggregate tables contain summarized statistics aggreated over different time periods. The use of aggregate tables gives the Open XDMoD portal fast query times, but it has the disadvantage of only being able to support predefined queries on the data. Open XDMoD also supports access to the individual records in the fact tables (known as the raw data), via the *Data Export* tab. When this tab was designed, there was concern that long running SQL queries would impact the responsiveness of the portal; thus, the tab was designed to service export requests in batch mode run nightly, meaning users usually need to wait to download data after requesting it. For the Python API, we decided to make it possible to obtain the same data programmatically and in real time through a method called `get_raw_data()`. This method takes `duration` and `realm` parameters to specify which data to retrieve. Specific fields (columns) of data can be requested via a `fields` parameter, and data can be limited to certain values using the `filters` parameter. Because the fact tables have many more rows of data than the aggregate tables, requests for raw data take a noticeably longer time, so we added a parameter to `get_raw_data()` called `show_progress` that specifies whether to periodically print how many rows have been obtained so far. An example of the `get_raw_data()` method is shown below, obtaining three days' worth of raw data about compute jobs, specifically the wall time elapsed and memory used by each finished job for projects in the field of Chemical Engineering. The resulting data frame contains a column for wall time, a column for memory used, and a row for each job.

```
raw_data = data_warehouse.get_raw_data(
    duration=('2023-05-01', '2023-05-03'),
    realm='Jobs',
    fields=(
        'Wall Time',
        'Memory Used'
    ),
    filters={
        'Field of Science': 'Chemical Engineering'
    },
    show_progress=True
)
```

We wanted the initial version of the API to be as simple as possible, using a minimal set of methods and parameters needed to specify the data to obtain from the data warehouse. Using just the `get_data()` and `get_raw_data()` methods, the user is able to obtain any of the data from the aggregate and fact tables in any of the realms that the access controls allow them to view. In order for the API user to obtain lists of the options available for the various parameters of these two methods, we also provided a set of methods named `describe_realms()`, `describe_raw_realms()`, `describe_metrics()`, `describe_dimensions()`, `describe_raw_fields()`, `get_filter_values()`, `get_durations()`, and `get_aggregation_units()`. These methods return data frames containing IDs, names, and/or descriptions of the various realms, metrics, dimensions, fields, etc.

We wanted the Python API to be easily installed via common mechanisms for installing Python packages. Thus, we created a project called `xdmod-data` on the Python Package Index (PyPI) [22] so that users could install it using the command `pip install xdmod-data`. We numbered the initial version 1.0.0 with the intent that the version numbering would follow Semantic Versioning [23].

Jupyter Notebook Development

In order to provide examples showing how to use the Python API, we created a set of Jupyter notebooks. Jupyter notebooks are interactive documents that can be run in web browsers and which contain a mix of documentation and code that can be run in real time. The notebooks can function like an interactive textbook by providing explanations of code next to the code itself, allowing the user to run examples and see results without needing to switch to another application or runtime environment. Jupyter notebooks can also be exported as PDF or interactive HTML documents that can be shared or run offline. Jupyter supports a wide range of languages; primarily Julia [16], Python, and R [14] (the name Jupyter being a portmanteau of these three language names), which are all popular languages used for data science and machine learning.

We decided to develop an initial set of three Jupyter notebooks to demonstrate use of the Data Analytics Framework. The first example notebook is meant both to demonstrate how to make charts like those available in the Open XDMoD portal and to enhance the analysis and charting beyond what is available in the portal. The second notebook demonstrates the new functionality for retrieving raw data, and the third notebook shows an example of using machine learning to analyze raw data. In all three notebooks, we included a code cell that installs or upgrades the `xdmod-data` Python package; in this way, we could ensure that the same Python environment was used to install the package and run the example code in the notebook. We also included code cells that create a file for storing the user's API token (used for authentication and discussed further in Sect. 3.4.1) and that read the contents of the file into the environment, to encourage keeping the API token secure and separate from the notebook, since the notebooks are encouraged to be shared, while the token should be kept secure. The notebooks also contain Markdown tables that show the data from the methods `describe_realms()`, `describe_metrics()`, etc., so that each of the methods of the API can be documented.

We decided that the example notebooks would use the Plotly [24] Python library to make charts, which is consistent with a concurrent project of the Open XDMoD team to convert the existing portal charts from rendering with the HighCharts [25] JavaScript library to the Plotly library. We created a Python module within the framework called `themes` for setting the style of the Plotly charts to use the same colors and fonts as those in the portal. We included charts in the notebooks that could be made through the portal and charts that could not (e.g., a box and whiskers plot).

The notebooks are meant to be extended and adapted, and users are able to use plotting libraries besides Plotly. Because the data are available in Pandas data frames, users can use any Python plotting library they prefer for charting the data, and users can also export the data for use in other programming languages or analysis environments.

Open XDMoD Portal Development

The Open XDMoD portal backend consists of PHP source code, MariaDB databases, and configuration files in INI and JSON formats. The frontend user interface is coded in HTML, CSS, and JavaScript. User interactions on the portal cause HTTP requests to be made to PHP endpoints in the backend that interact with the data warehouse using MariaDB. For the Data Analytics Framework, we made some changes to the backend and frontend to support programmatic access to the HTTP endpoints via the new Python

API. We also documented the HTTP endpoints using the OpenAPI specification [26].

API Token Authentication

The Open XDMoD HTTP endpoints use session-based authentication via cookies, which are obtained when the user logs in to the portal with a username and password. For convenient use of the Data Analytics Framework, we decided to add an API token authentication mechanism that would allow read-only access restricted only to certain endpoints and operations, namely only those endpoints needed for the operation of the Python API (e.g., obtaining aggregate data; lists of possible filter values; or descriptions of realms, metrics, and dimensions).

Users create API tokens for themselves through the portal, and once they have a token, they can provide it to the API as an environment variable. For simplicity, we decided that each user would have a single API token at a time. Each token is given an expiration date, defaulting to six months, and configurable by system administrators of the portal.

To enable the creation, reading, and deletion of API tokens, we added a new table to the data warehouse, a new interface component to the Open XDMoD portal, and a new HTTP endpoint. The interface component on the portal sends requests to the HTTP endpoint, and the HTTP endpoint makes the relevant database queries and sends responses back to the user interface, which displays helpful messages to the user.

New HTTP Endpoints for Raw Data

In addition to creating the new HTTP endpoint for manipulating API tokens and modifying existing endpoints to enable API token authentication, we also created two new endpoints for obtaining raw data. As mentioned earlier, raw data could previously only be obtained as a batch process that ran once per day. In order to support real-time requests for raw data that also include filtering, we created a new HTTP endpoint called `raw-data`. The number of rows that can be obtained in a single request to this endpoint is defined to be 10,000 by default and is configurable by the system administrator in an INI file. In order for this maximum number to be obtained programmatically, we also created an HTTP endpoint called `raw-data/limit` that simply returns the number.

For the Python API to obtain raw data, it first makes a request to the `raw-data/limit` endpoint to obtain the configured maximum number. Then, it makes a request to the `raw-data` endpoint, specifying the realm, date range, and any specific fields and/or filters. Next, it checks if the number of rows it received is the configured maximum number. If it is, it makes the same request again, specifying an offset equal to the number of rows it has received so far. If it isn't, it makes no further requests. By making requests in this way, the API is able to iteratively receive all the data, with opportunities along the way to give feedback to the user on the number of rows obtained so far (if the `show_progress` parameter is set). This splitting up of each request into multiple smaller requests reduces the size of queries that are made to the data warehouse, reducing the risk of Denial of Service. By providing an offset parameter for each of the smaller requests, the state of the overall request does not need to be maintained by the server.

Version Control, Testing, and Documentation

To track changes made to the Open XDMoD source code, we utilized the existing `xdmod` GitHub repository [27]. We created new GitHub repositories to track changes to the Python API source code [28] and example Jupyter notebooks [29]. Using GitHub repositories allows us to be trasparent about our development and facilitate contributions from the community. These repositories include automated continuous integration testing through the CircleCI software [30], helping us better identify bugs as new changes are integrated into the code. We developed automated tests of the Python code using the pytest library [31], and we extended existing integration tests of the PHP portal code using the PHPUnit library [32]. Our integration tests ran on the CentOS 7 and Rocky 8 Linux operating systems, which are the two operating systems we officially support in version 10.5 of Open XDMoD.

In addition to the documentation in the Jupyter notebooks, we also developed documentation in the Open XDMoD portal's user guide to explain how users can generate API tokens, and on the Open XDMoD website to inform system administrators how to configure their installations of Open XDMoD to use the Data Analytics Framework.

Version 1.0.0 of the Framework

We completed and released version 1.0.0 of the Data Analytics Framework for XDMoD on July 21, 2023, to the `xdmod-data` GitHub repository [28] and the Python Package Index (PyPI) [22]. We upgraded the ACCESS-hosted version of Open XDMoD to a new 10.5 version on July 19, 2023. Users with ACCESS accounts can log in to the ACCESS XDMoD portal, generate an API token, and use it to run the Python API to obtain data from the ACCESS-hosted version of Open XDMoD. We released version 10.5 of Open XDMoD on September 11, 2023, which allows portal users of centers with Open XDMoD installations to generate API tokens and use the Python API to obtain data from that center's Open XDMoD data warehouse.

The example Jupyter notebooks are available for download from the `xdmod-notebooks` GitHub repository [29]. This repository contains instructions for installing either the Anaconda [33] or Docker [34] software to run the notebooks. Each notebook contains a code cell that installs the `xdmod-data` Python package, or the package can be installed directly from PyPI using the command `pip install xdmod-data`.

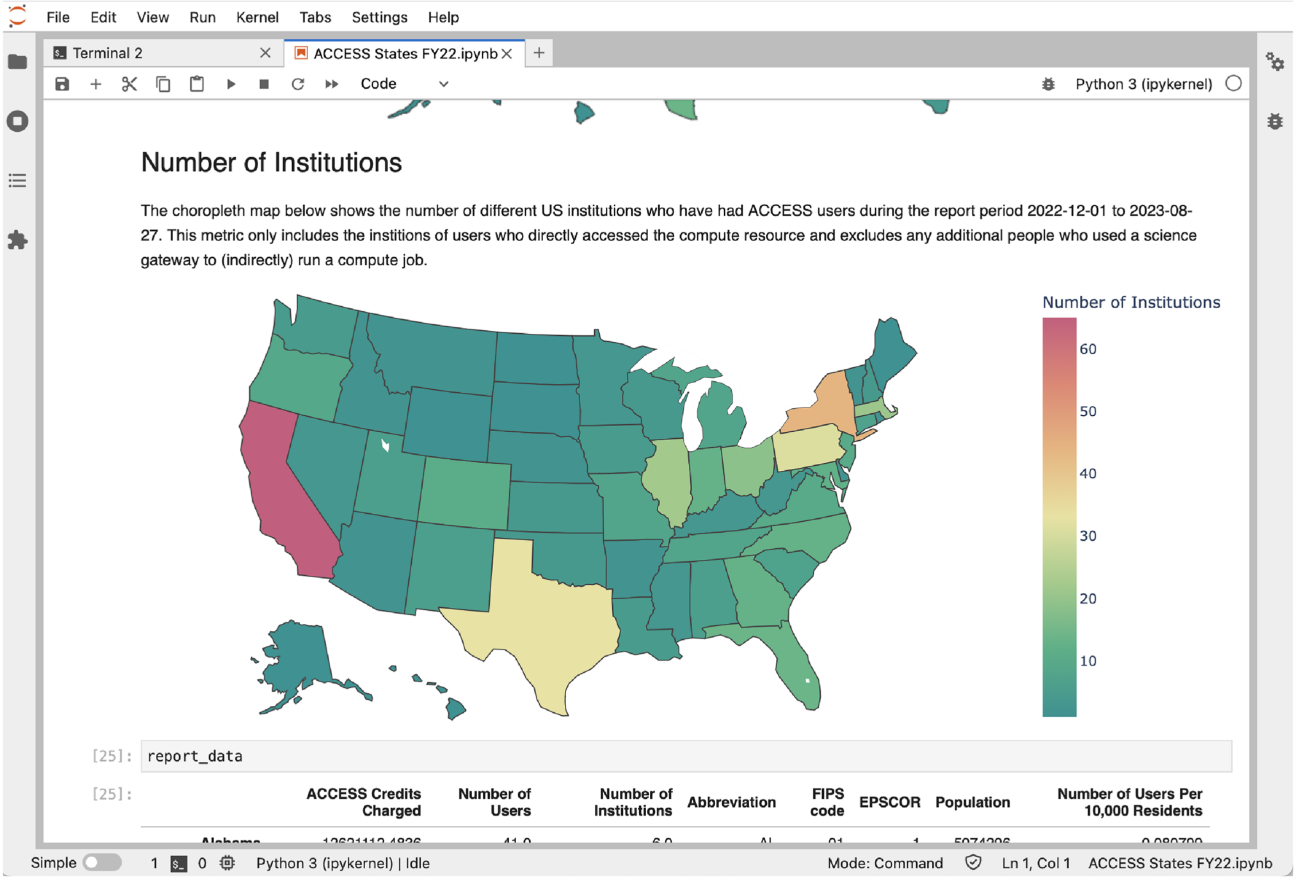
We welcome contributions from the community to the GitHub repositories in the form of GitHub Pull Requests, in which contributors make changes and submit a request for review by our team.

Case Studies

In this section we detail a couple of example case studies demonstrating use of the Data Analytics Framework.

CI Usage Reporting

Our team has begun using the Data Analytics Framework to report on ACCESS usage to the NSF. For example, we created a Jupyter notebook in which we reported the usage of ACCESS-allocated compute resources by US state over a nine month period. We used the Python API to obtain three metrics from the ACCESS-hosted version of Open XDMoD: 1) total number of ACCESS Credits charged, 2) number of active users, and 3) number of active institutions. The data were obtained from the *Jobs* realm, grouped by the dimension *User Institution State*, and filtered only to include results where the dimension *User Institution Country* is the United States. We also joined in data from other sources; for example, whether the state is an EPSCoR [35] state, and the state's population (used to calculate the number of users of ACCESS-allocated resources per 10,000 residents). We output the data frame as a Markdown table and created a choropleth plot using Plotly. A screenshot of the notebook is shown in Fig. 3. This notebook, along with other reporting on the usage of ACCESS-allocated resources, is planned to be



Number of Institutions

The choropleth map below shows the number of different US institutions who have had ACCESS users during the report period 2022-12-01 to 2023-08-27. This metric only includes the institions of users who directly accessed the compute resource and excludes any additional people who used a science gateway to (indirectly) run a compute job.

```
[25]: report_data
```

[25]:

	ACCESS Credits Charged	Number of Users	Number of Institutions	Abbreviation	FIPS code	EPSCOR	Population	Number of Users Per 10,000 Residents
Alabama	12621112.4836	41.0	6.0	AL	01	1	5074296	0.080799

Fig. 3 Screenshot of an analytic notebook containing a choropleth map that shows the number of different US institutions with ACCESS users during the report period 2022-12-01 to 2023-08-27

made available to the public in the ACCESS Knowledge Base [36].

Machine Learning Classification

As another example of the type of analysis one can do with the Data Analytics Framework, and to provide a template for how to use it, we created a Jupyter notebook for a case study of a random forest classification, a widely used machine learning algorithm. The goal of the study was to see if available characteristics of running compute jobs could be used to predict the software application that was being run. The Python API was used to fetch two months (January 1, 2023, through February 28, 2023) of job-level performance data from jobs running on the Bridges-2 resource from Pittsburgh Supercomputing Center. Job-level performance data is available through the *SUPREMM* realm. The ACCESS MMS team populates this realm by receiving node-level performance data from ACCESS Resource Providers who collect data by running software such as Performance Co-Pilot [37] or Prometheus [38] (or other software whose data can be exported into a common format) on their resources' compute nodes. The ACCESS MMS team summarizes the node-level data into job-level data using the SUPReMM [39] software. The job-level data are then ingested into the *SUPREMM* realm, making them available for analysis through the Data Analytics Framework and the portal. Centers running their own installations of Open XDMoD can also collect node-level performance data from their own compute nodes, run the SUPReMM software to summarize the job-level performance data, and ingest the data into the *SUPREMM* realm of their installations of Open XDMoD.

In this case study, nine fields from the raw data were used as predictors:

- *Wall Time*: the wall-clock duration of the job,
- *CPU User*: the ratio of user CPU time to total CPU time for the cores that the job was assigned;
- *Total memory used*: the total memory used by the OS on all the nodes, including the page and buffer caches;
- *Net Ib0 Rx*: the average number of bytes received by the InfiniBand network interface on each node;
- *Net Ib0 Tx*: the same as *Net Ib0 Rx* but for bytes transmitted instead of received;
- *CPU User cov*: the coefficient of variation for *CPU User*;
- *Memory Used Cov*: the coefficient of variation for *Memory Used*;
- *Net Ib0 Rx Cov*: the coefficient of variation for *Net Ib0 Rx*; and
- *Net Ib0 Tx Cov*: the coefficient of variation for *Net Ib0 Tx*.

We cleaned the data to remove jobs for which the application data were unavailable, jobs for which the application had been categorized in a group of applications rather than as a single application (e.g., applications run via Python, R, or shell scripts), and jobs for which the application's license prohibits performance reporting. We selected only jobs running the eight most common applications for the given time range. The data were split 90/10 into 40,134 training rows and 4,460 test rows. We used the random forest algorithm from the Python scikit-learn library [40]. The model produced an out-of-bag accuracy of 97%. The top predictors based upon the variable importance were *CPU User cov*, *CPU User*, *Wall Time*, *Net Ib0 Tx*, *Total memory used*, and *Net Ib0 Rx*, with very low variable importance for *Net Ib0 Rx Cov*, *Net Ib0 Tx Cov*, and *Memory Used Cov*. The confusion matrix is shown in Fig. 4. The Jupyter notebook for this case study is provided in the `xdmod-notebooks` repository [29].

Conclusion

The initial implementation of the Data Analytics Framework meets our goals of providing programmatic API access to the data in Open XDMoD, overcoming limitations of the Open XDMoD portal, and enabling a wide variety of analyses and custom reporting that can make use of commonly available tools and environments. In this section we discuss the potential impact of the framework and planned future work.

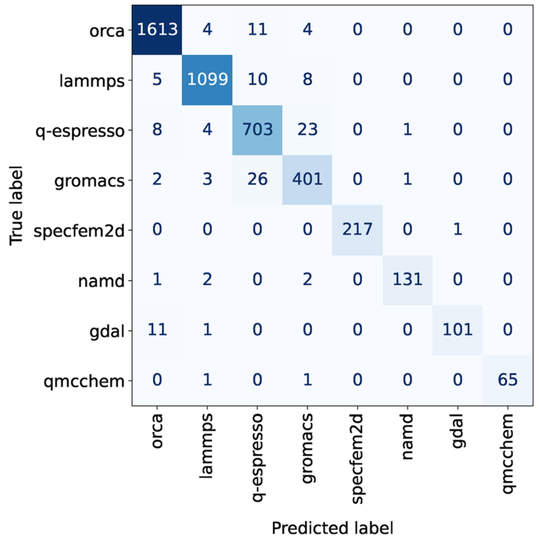


Fig. 4 Confusion Matrix for the application classification case study. Applications were predicted with an accuracy of 97%

Potential Impact

Some initial types of use cases that we can foresee for the Data Analytics Framework are (1) CI studies: the Data Analytics Framework provides information that can lend insight into which CI was used, how it was used, and the opportunity to extrapolate what will be needed in the future; (2) User studies: how users interact with the CI, how the usage is evolving, and how the CI can be improved to better serve the users; (3) Return on Investment (ROI) studies: with funding and usage data in the Data Analytics Framework, there is the basis for ROI models and analyses. Of course, the depth and breath of the data available in the framework will provide many opportunities for a wide variety of bespoke reports, models, and analyses.

We have received feedback from users who anticipate the framework will be useful for generating reports of their centers' usage of CI, in particular, when reports need to be generated in a way that is more flexible than can currently be provided by the Open XDMoD portal. Jupyter notebooks can also be adapted by CI centers for their support and outreach. For example, when engaging with users, support staff can provide a Jupyter notebook that highlights some of the Open XDMoD data pertinent to those particular users' use of the CI resources provided by the center.

As previously described, our team is using the framework to report to the NSF on resource utilization of ACCESS-allocated resources. As the team expands its monitoring and measurement of the national CI ecosystem, the Data Analytics Framework will be a key piece in our analysis of data from resources provided across the portfolio of NSF programs as well as individual centers. As an example, our team is developing models of cyberinfrastructure usage based on the data from these sources, which can potentially be used to categorize and predict resource usage patterns and needs.

Programmatic analysis of detailed job-level performance data mapped to applications, users, disciplines, and resources allows for more comprehensive understanding of usage and efficiency patterns, which can help assess how well current CI systems are responding to researcher needs and to inform future CI procurement and investment at the local and national levels. For example, the framework can be used to analyze uptake and efficient utilization of new resources designed for artificial intelligence and machine learning applications, focusing in on particular disciplines as desired.

Future Work

We plan to continuously improve the Data Analytics Framework and periodically release new versions. We currently have a list of changes that include new features and bug fixes; some of these are discussed in Sect. 5.2.1.

When new features are added to the API, we will create new notebooks that document examples of using these features. We also plan to make new notebooks that demonstrate different use cases of the framework. We encourage the community to contribute modifications and additional notebooks that we will curate via the `xdmod-notebooks` GitHub repository.

Future additions are planned for Open XDMoD that will expand the realms of cyberinfrastructure data collected and reported. We plan to expand the collection and reporting of networking data, storage data, and Open OnDemand [41] usage. As new resources are integrated into ACCESS, data on these resources will be available through the Open XDMoD portal and the Data Analytics Framework.

Improvements to the API

We plan to improve performance of queries to the data warehouse, in particular queries for raw data. We are aware of certain improvements that we can make in how the queries are constructed, and we will further investigate other potential performance improvements.

We plan to increase the logging capabilities of the framework to be able to better understand who is using it and for what purposes. This will help us better report on the impact of the framework and focus our improvements to it.

In version 1.0.0 of the framework, the filters for raw data are incomplete. The raw data queries for some realms are not properly joining the aggregate tables to the fact tables. Some raw data fields also have inconsistent labels between the fact and aggregate tables. In a future version, we will complete the filtering and clean up labels so they are consistent.

Currently both the `get_data()` and `get_raw_data()` methods do not have a way to limit queries to a certain number of results. In particular for `get_raw_data()`, if only a certain number of rows are needed, it can have a noticeable effect on performance to request all the rows rather than limiting them to just the number needed. In a future version of the framework, we plan to add a `limit` parameter.

There are use cases in which multiple metrics are desired for the same realm, time range, dimension, and filters. Currently this is achieved by making multiple calls to `get_data()`. In a future version we will enable multiple metrics to be specified in a single call to `get_data()`, simplifying the code and possibly improving on query execution time.

Based on feedback received from initial users, we plan to make the methods have more consistent names. For example, there is currently an inconsistency in some method names starting with `get_` and others starting with `describe_`. To make it easier to remember names of methods, we plan to add `get_` aliases for the `describe_` methods;

for example, `get_dimensions()` will be an alias for `describe_dimensions()`.

There is also some interchangeable terminology used within the Open XDMoD portal, such as metrics being called "statistics" or dimensions being called "group bys," so in a future version, we plan to make parameter names in the API be able to be specified using any of the interchangeable terms.

Currently the API is documented through the example Jupyter notebooks and code comments in the source code of the `xdmod-data` package. The code comments can be converted to HTML using the Sphinx Python library [42], and we plan to use this library to produce detailed documentation of the API that will be hosted on the Open XDMoD website.

We also plan to improve the automated test coverage of the code with the help of a library like coverage.py [43].

Hosted Jupyter Notebooks

One of the strengths of the Open XDMoD portal is that it is usable by anyone with a web browser without the need to install any specialized software. The initial version of the Data Analytics Framework does require local tool installation, however, to run the Jupyter notebooks. Even though we are using common tools that are widely used in the community, this tool requirement results in a barrier to entry to use the framework. We plan to make it possible for users to run the framework through Jupyter notebooks directly in a web browser without needing to download software. We intend to enable this through existing production software such as JupyterHub [44] or Open OnDemand [41].

The *Data Export* tab on the Open XDMoD portal will eventually be deprecated in favor of the Data Analytics Framework. We also plan to work towards replacing the existing *Report Generator* tab with a Jupyter-based interface that uses the framework. We plan to include Jupyter notebooks as report templates and provide a periodic reporting capability such as what currently exists in the portal.

R Version

We plan to provide example Jupyter notebooks showing how to interact with the framework using the R language. We will do this starting with the R package called Reticulate [45], which allows embedding of Python code in R notebooks. Because the Python API obtains data and stores them in Pandas data frames, it will be straightforward to convert the data frames into R data frames. After prototyping this implementation, we will decide how we can provide more comprehensive R support.

Outreach

We are planning outreach activities to make the community aware of how they can use the framework to analyze the data from their own centers as well as the historical data from resources allocated through TeraGrid, XSEDE, and ACCESS. Our team presented a tutorial showing how to use the framework at the PEARC '23 conference. We are interested in holding additional tutorials and mini-workshops for specific audiences. We would also like to make a video tutorial that can be updated as changes are made to the framework.

Appendix A Metrics in Open XDMoD

The default metrics available in each realm of Open XDMoD are listed in the following sections. Metrics in Open XDMoD can be grouped by time units of day, month, quarter, or year; and they can be grouped and filtered by the dimensions in Appendix B.

A.1 Metrics in *Jobs* and *Gateways* realms

The *Jobs* and *Gateways* realms have metrics for the total and per-job counts of CPU hours, GPU hours, node hours, wait hours, and wall hours; numbers of jobs submitted, started, running, and ended; numbers of active users, principal investigators, and resources; the number of GPUs per job; the CPU core hour utilization percentage; and numbers of CPU cores per job: total, minimum, maximum, average, normalized average, average weighted by CPU hours, and average weighted by GPU hours.

A.2 Metrics in *SUPREMM* realm

The *SUPREMM* (job-level performance) realm has metrics for the total and per-job numbers of wait hours, wall hours requested, and wall hours used; the wall time accuracy; the numbers of jobs submitted, started, running, and ended; the total numbers and average percentages weighted by node hour or GPU hour of CPU hours and GPU hours (allocated and utilized); the total numbers and average percentages weighted by core hour of CPU hours spent in system, user, and idle states; the averages per core weighted by core hour for the number of cycles per instruction, the ratio of clock ticks to L1D cache loads, the floating point operations per second, the core imbalance, the memory, the maximum memory, and the memory bandwidth; and the averages weighted by node hour for

L1D load homogeneity, InfiniBand rate, mount point and block device read and write rate, and filesystem receive and transmit rate.

A.3 Metrics in *Cloud* realm

The *Cloud* realm has metrics for number of sessions started, active, and ended; total and per-session wall hours; total CPU hours; CPU core hour utilization percentage; and metrics weighted by wall hours for average cores, memory, and root volume storage reserved.

A.4 Metrics in *Storage* realm

The *Storage* realm has metrics for logical and physical usage in bytes; hard and soft quota thresholds; numbers of users and files; and logical quota utilization percentage.

A.5 Metrics in *OnDemand* realm

The *OnDemand* realm has metrics for the numbers of page loads, sessions (total and per user), active users, and Open OnDemand applications.

Appendix B Dimensions in Open XDMoD

Metrics in Open XDMoD can be grouped and filtered by the following dimensions.

B.1 Common dimensions

Dimensions common to all the realms include users, system usernames, principal investigators, resources, resource types, service providers, and a three-level hierarchy that by default contains fields of science, parent sciences, and NSF directorates. This hierarchy can be customized by each center that runs an installation of Open XDMoD, e.g., to use decanal units, departments, and groups/accounts instead.

B.2 Additional dimensions in *Jobs* realm

The *Jobs* realm adds dimensions for job sizes, GPU counts, wall times, quality of service categories, and queues.

B.3 Additional dimensions in *SUPREMM* realm

The *SUPREMM* (job-level performance) realm adds many of the same dimensions as the *Jobs* realm as well as dimensions for applications; data sources; exit statuses; share modes; homogeneity and catastrophe ranks; and values of wall time accuracy, CPU and GPU utilization, cycles per instruction, peak memory usage, and InfiniBand and filesystem receive rates.

B.4 Additional dimensions in *Cloud* realm

The *Cloud* realm adds dimensions for virtual machine instance types, states, core sizes, memory sizes, submission venues, domains, and projects.

B.5 Additional dimension in *Storage* realm

The *Storage* realm adds a dimension for mount points.

B.6 Additional dimensions in *Gateways* realm

The *Gateways* realm adds dimensions for gateways and gateway acronyms.

B.7 Additional dimensions in *OnDemand* realm

The *OnDemand* realm adds dimensions for browsers, operating systems, geographical locations, and Open OnDemand applications and usernames.

Appendix C Additional metrics and dimensions in ACCESS-hosted version of Open XDMoD

The ACCESS-hosted version of Open XDMoD adds additional metrics and dimensions beyond those available by default in Open XDMoD.

C.1 Metrics

The ACCESS-hosted version of Open XDMoD adds metrics for numbers of active allocations and institutions, allocation usage rates, and utilization statistics normalized across different resources.

C.2 Dimensions

The ACCESS-hosted version of Open XDMoD adds dimensions for allocations, resources, and information about users' and principal investigators' institutions (name, state, and country).

Acknowledgements This material is based upon work supported by the National Science Foundation under Grant No. OAC 2137603. Nick Cruz assisted with initial prototyping of the Python and Jupyter implementations. The rest of the ACCESS MMS team (Matthew D. Jones, Abani K. Patra, Gregary Dean, Andrew Stoltman, Joshua Furlani, Jennifer Schopf, Shava Smallen, Vipin Chaudhary, Zahra Rahmani, Stephen Harrell, and Matt Cawood) provided additional support, as did the other ACCESS teams.

Author Contributions Thomas R. Furlani, Robert L. DeLeon (RLD), and Joseph P. White (JPW) conceptualized the Data Analytics Framework and provided supervision over its design and implementation. JPW wrote initial prototypes for the various pieces of the framework (Python API, example Jupyter notebooks, and new HTTP endpoints). Aaron Weeden (AW) completed these implementations and added automated testing, documentation, and software packaging via PyPI and GitHub. JPW designed and implemented the case study of ACCESS usage reporting. RLD designed and implemented the case study of random forest machine learning classification, with refinements from AW. Ryan Rathsam designed and implemented the API token authentication. Nikolay A. Simakov developed a prototype example R notebook that uses the Data Analytics Framework and the Reticulate library. Conner Saeli provided quality assurance testing and feedback.

Funding This material is based upon work supported by the National Science Foundation under Grant No. OAC 2137603.

Data availability The data used in the Case Studies section and Figs. 3–4 came from ACCESS XDMoD (https://xdmod.access-ci.org/). The CI Usage Reporting case study used additional data: the list of NSF EPSCoR jurisdictions (https://new.nsf.gov/funding/initiatives/epscor/state-websites) and population data from the U.S. Census ("Annual Estimates of the Resident Population for the United States, Regions, States, District of Columbia and Puerto Rico: April 1, 2020 to July 1, 2022 (NST-EST2022-POP)"—https://www.census.gov/data/tables/time-series/demo/popest/2020s-state-total.html#v2022).

Declarations

Conflict of Interest On behalf of all authors, the corresponding author states that there is no Conflict of interest.

References

1. Anderson G, Jankowski J, Boroush M. U.S. R &D Increased by $51 Billion in 2020 to $717 Billion; Estimate for 2021 Indicates Further Increase to $792 Billion. National Center for Science and Engineering Statistics (NCSES), National Science Foundation. NSF 23-320. Available from: https://ncses.nsf.gov/pubs/nsf23320. Accessed 29 Sept 2023.
2. Boerner TJ, Deems S, Furlani TR, Knuth SL, Towns J. ACCESS: Advancing Innovation: NSF's Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support. In: Practice and Experience in Advanced Research Computing. PEARC '23. New York, NY, USA: Association for Computing Machinery; 2023. p. 173–176. Available from: https://doi.org/10.1145/3569951.3597559.
3. National Science Foundation. Campus Cyberinfrastructure (CC*). https://new.nsf.gov/funding/opportunities/campus-cyberinfrastructure-cc/. Accessed 20 Sept 2023.
4. National Science Foundation. Cyberinfrastructure for Sustained Scientific Innovation (CSSI). https://new.nsf.gov/funding/opportunities/cyberinfrastructure-sustained-scientific. Accessed 20 Sept 2023.
5. Towns J, Cockerill T, Dahan M, Foster I, Gaither K, Grimshaw A, et al. XSEDE: accelerating scientific discovery. Comput Sci Eng. 2014;16(5):62–74. https://doi.org/10.1109/MCSE.2014.80.
6. Catlett C, Allcock WE, Andrews P, Aydt R, Bair R, Balac N, et al. Teragrid: Analysis of organization, system architecture, and middleware enabling new types of applications. IOS press; 2008.
7. Palmer JT, Gallo SM, Furlani TR, Jones MD, DeLeon RL, White JP, et al. Open XDMoD: a tool for the comprehensive management of high-performance computing resources. Comput Sci Eng. 2015;17(4):52–62. https://doi.org/10.1109/MCSE.2015.68.
8. Kimball R, Ross M. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. 3rd ed. Wiley Publishing; 2013.
9. Jones MD, White JP, Innus M, DeLeon RL, Simakov N, Palmer JT, et al. Workload Analysis of Blue Waters. ArXiv e-prints. 2017;arXiv:1703.00924.
10. Simakov NA, White JP, DeLeon RL, Gallo SM, Jones MD, Palmer JT, et al. A Workload Analysis of NSF's Innovative HPC Resources Using XDMoD. ArXiv e-prints. arXiv:1801.04306 (2018).
11. White JP, DeLeon RL, Furlani TR, Gallo SM, Jones MD, Ghadersohi A, et al. An Analysis of Node Sharing on HPC Clusters Using XDMoD/TACC_Stats. In: Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment. XSEDE '14. New York, NY, USA: ACM; 2014; p. 31:1–31:8. Available from: https://doi.org/10.1145/2616498.2616533.
12. White JP, Innus M, Deleon RL, Jones MD, Furlani TR. Monitoring and Analysis of Power Consumption on HPC Clusters Using XDMoD. In: Practice and Experience in Advanced Research Computing. PEARC '20. New York, NY, USA: Association for Computing Machinery; 2020; p. 112–119. Available from: https://doi.org/10.1145/3311790.3396624.
13. Scrivner O, Singh G, Bouchard SE, Hutcheson SC, Fulton B, Link MR, et al. XD Metrics on demand value analytics: visualizing the impact of internal information technology investments on external funding, publications, and collaboration networks. Front Res Metr Anal. 2018. https://doi.org/10.3389/frma.2017.00010.
14. R Core Team. R: A language and environment for statistical computing. Vienna, Austria. Available from: https://www.R-project.org/.
15. Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy. Nature. 2020;585(7825):357–62. https://doi.org/10.1038/s41586-020-2649-2.
16. Bezanson J, Edelman A, Karpinski S, Shah VB. Julia: a fresh approach to numerical computing. SIAM Rev. 2017;59(1):65–98. https://doi.org/10.1137/141000671.
17. RStudio Team. RStudio: Integrated Development Environment for R. Boston, MA. Available from: http://www.rstudio.com/.
18. Kluyver T, Ragan-Kelley B, Pérez F, Granger B, Bussonnier M, Frederic J, et al. Jupyter Notebooks - a publishing format for reproducible computational workflows. In: Loizides F, Schmidt B, editors., et al., Positioning and power in academic publishing: players, agents and agendas. IOS Press; 2016. p. 87–90.
19. The MathWorks, Inc. MATLAB. https://www.mathworks.com/products/matlab.html. Accessed 20 Sept 2023.
20. McKinney W. Data structures for statistical computing in python. In: Stéfan van der Walt, Jarrod Millman, editors. Proceedings of the 9th Python in Science Conference; 2010. p. 56 – 61.
21. Reitz K. Requests: HTTP for Humans^{TM}. https://requests.readthedocs.io/en/latest/. Accessed 20 Sept 2023.
22. Python Software Foundation. Python package index - PyPI. https://pypi.org/. Accessed 20 Sept 2023.
23. Preston-Werner T. Semantic versioning. http://semver.org/. Accessed 20 Sept 2023.
24. Plotly. Plotly open source graphing library for python. https://plotly.com/python/. Accessed 20 Sept 2023.
25. Highcharts. Highcharts. https://highcharts.com/. Accessed 20 Sept 2023.

26. SmartBear Software. OpenAPI specification. https://swagger.io/specification/. Accessed 20 Sept 2023.
27. University at Buffalo Center for Computational Research. Open XDMoD. GitHub. https://github.com/ubccr/xdmod. Accessed 20 Sept 2023.
28. University at Buffalo Center for Computational Research. xdmod-data. GitHub. https://github.com/ubccr/xdmod-data. Accessed 20 Sept 2023.
29. University at Buffalo Center for Computational Research. xdmod-notebooks. GitHub. https://github.com/ubccr/xdmod-notebooks. Accessed 20 Sept 2023.
30. Circle Internet Services, Inc. CircleCI. https://circleci.com/. Accessed 20 Sept 2023.
31. Krekel H, pytest-dev team. pytest. https://docs.pytest.org/en/7.4.x/. Accessed 20 Sept 2023.
32. Bergmann S. PHPUnit. https://phpunit.de/. Accessed 20 Sept 2023.
33. Anaconda Inc. Anaconda. https://www.anaconda.com/. Accessed 20 Sept 2023.
34. Docker Inc. Docker. https://www.docker.com/. Accessed 20 Sept 2023.
35. National Science Foundation. Established program to stimulate competitive research (EPSCoR). https://new.nsf.gov/funding/initiatives/epscor. Accessed 20 Sept 2023.
36. ACCESS Support. ACCESS knowledge base. https://support.access-ci.org/knowledge-base. Accessed 20 Sept 2023.
37. Silicon Graphics Inc, Aconex, and Red Hat. Performance co-pilot. https://pcp.io/index.html. Accessed 20 Sept 2023.
38. Prometheus Authors. Prometheus. Accessed: 2023-09-29. Available from: https://prometheus.io/.
39. University at Buffalo Center for Computational Research. SUPReMM. GitHub. https://github.com/ubccr/supremm. Accessed 20 Sept 2023.
40. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: machine Learning in Python. J Mach Learn Res. 2011;12:2825–30.
41. Hudak D, Johnson D, Chalker A, Nicklas J, Franz E, Dockendorf T, et al. Open OnDemand: a web-based client portal for HPC centers. J Open Source Softw. 2018;3(25):622. https://doi.org/10.21105/joss.00622.
42. The Sphinx developers. Sphinx. https://www.sphinx-doc.org/en/master/. Accessed 20 Sept 2023.
43. Batchelder N. coverage.py. https://coverage.readthedocs.io/en/7.2.7/. Accessed 20 Sept 2023.
44. Project Jupyter. JupyterHub. https://jupyter.org/hub. Accessed 20 Sept 2023.
45. Kalinowski T, Ushey K, Allaire J, RStudio, Tang Y. R Interface to python. https://rstudio.github.io/reticulate/. Accessed 20 Sept 2023.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.