MDSplusML - Optimizations for data access to facilitate machine learning pipelines

Joshua Stillerman[a], Stephen Lane-Walsh[a], Mark Winkel[a], Cristina Rea[a], Gregorio Luigi Trevisan[a], Aleksandar Jelenak[b], John Readey[b]

[a]*Massachusetts Institute of Technology Plasma Science and Fusion Center*
[b] *HDF group*

The MDSplus[1-2] data management system is widely used in the magnetic fusion energy research community for data storage, management, and remote access. The system provides data access through a vector based, interpreter API. It was developed and optimized for rapid single shot analyses. Machine Learning applications require data from large numbers of shots and potentially from different experimental devices. We are developing tools to enable the rapid retrieval of limited sets of data from large numbers of shots. The system will cache the requested quantities in a data warehouse overnight, and be able to quickly provide them as inputs to machine learning tasks. The cache will eventually be both transparent and extensible. At this time, various caching mechanisms are being tested and benchmarked using the queries for approximately 100 quantities that are typically used by disruption-warning ML workflows. The performance of various caching schemes varies greatly depending on the environment they are deployed in. We provide comparisons of the performance of native MDSplus, HSDS[3] cache, and mongodb[4] cache in various environments. The end goal is to provide fast data access to commonly queried quantities regardless of the environment.

Keywords: Data Management, Machine Learning, Magnetic Fusion

1. Introduction

The magnetic fusion energy research community has long relied on MDSplus as a robust data management system for storage, management, and remote access of experimental data. MDSplus provides a loosely typed vector-based interpreter API that has been optimized for rapid single-shot analyses and long-term data storage. However, the emergence of machine learning (ML) applications in fusion research has introduced new challenges and requirements for data access and management.

This paper presents ongoing work to develop tools and optimizations for MDSplus to enable rapid retrieval of data from large numbers of shots, potentially spanning different experimental devices. These developments aim to bridge the gap between traditional shot-by-shot data analysis and the data-intensive needs of machine learning pipelines.

2. Background and Motivation

2.1 MDSplus Overview

MDSplus has been a cornerstone in fusion research data management since its introduction in 1990. It offers a flexible and powerful system for handling the complex data structures and large volumes of information generated by fusion experiments. Users can annotate their data with rich metadata, allowing the data to retain its usefulness over long periods of time and through changes of personnel. The system's strength lies in its ability to provide rapid access to data for single-shot analyses, which has been the primary mode of operation in traditional fusion research. Much of the data from magnetic fusion experiments is either stored and managed by MDSplus, or accessible through the MDSplus remote data access layer.

2.2 Evolving Requirements

The advent of machine learning applications in fusion research has shifted the paradigm of data usage. While traditional analyses focus on in-depth examination of individual experimental shots, ML approaches often require access to data from a large number of shots, sometimes spanning multiple experiments or devices. This shift necessitates a re-evaluation of data access strategies to accommodate several new requirements.

There is a need for the ability to efficiently retrieve selected signals across numerous shots. Unlike traditional shot-by-shot analysis, ML algorithms often require consistent data points from a wide range of experimental runs, making fast and efficient bulk data retrieval crucial.

Modern research environments demand support for both on-premises and cloud-based data access. As computational resources increasingly move to cloud platforms, data management systems must be flexible enough to operate effectively in both local and distributed environments.

There is a shift in focus from speed in accessing individual data points to overall throughput. Machine learning models often require large volumes of data, making the rate at which entire datasets can be processed more important than the speed of retrieving single data points.

There is an increasing emphasis on adherence to FAIR[5] (Findable, Accessible, Interoperable, Reusable) principles. As the fusion research community becomes more interconnected and data-driven, ensuring that data is easily discoverable, accessible, compatible across different systems, and reusable for various purposes has become paramount.

These evolving requirements underscore the need for a more flexible and scalable approach to data access in

author's email: jas@mit.edu

fusion research, particularly as machine learning techniques become more prevalent in the field.

3. Design Considerations

3.1 User-Centric Approach

In developing optimizations for MDSplus, several key user requirements have been identified. These requirements reflect the needs and preferences of the research community and have shaped the design considerations for the MDSplusML project.

A primary consideration is that users prefer to perform their own feature extraction and data preparation. This preference for autonomy in data handling reflects the specialized nature of fusion research and the diverse analytical approaches employed by different researchers. Researchers require rapid retrieval of relevant data to efficiently set up their analyses and machine learning pipelines. Fast access to data is crucial. Since the users will do their own processing, processing pipeline tools do not need to be included in the system.

In order to provide scalability, user implemented data preparation is often performed in parallel. The tools must support both multi-threaded and multi-processed data access without degradation in performance or data integrity. Traditional shot by shot data analysis required fast transaction times for serial requests. Since the machine learning data preparation is usually similar across shots, it can often be done in parallel. This means that throughput as opposed to transaction time can be a more important factor in overall data processing speed. Of course, it is important that parallel processing does not lead to interference between threads.

Lastly, it is important to recognize that the cohort of machine learning researchers working on fusion data sets are often not the personnel who collected, and worked with these data on a day to day basis. They require more complete explanatory metadata so that they can find and understand their data inputs. In addition, the data access APIs that they expect tend to be non-fusion research specific. Providing APIs in these standard forms will both attract a wider set of users and allow them to leverage their non-domain specific technical skills.

These user-centric requirements collectively emphasize the need for a flexible system that provides scalable, fast access to measured quantities while allowing users to maintain control over their data preparation workflows. The MDSplusML optimizations aim to balance these needs, providing powerful tools for data access preserving while the autonomy and flexibility that researchers require.

3.2 FAIR Principles

Adherence to FAIR (Findable, Accessible, Interoperable, Reusable) principles is becoming increasingly important in scientific data management, particularly in the context of fusion research. The implementation of FAIR principles is motivated by the growing need to share and leverage data across a wider community of researchers, including those from outside the immediate fusion research field.

Inter-machine analyses are needed to create general models that will be useful for new and proposed machines. These require the use of a common ontology and representation of the quantities being used. The ITER experiment has developed Integrated Modelling & Analysis Suite (IMAS)[[6] for this purpose. While this may not be optimal and suit all the research community's preferences, it is being proposed for wide adoption. Each machine requires a mapping from their local naming and data representation to the shared definitions. Various tools using JSON to describe and implement the mapping are under development.

As fusion research progresses towards the goal of practical energy production, it is attracting interest from machine learning researchers who do not have intimate familiarity with the data and the problem space in general. This necessitates more explanatory metadata than has been previously needed by researchers working specifically in the field. Moreover, the increasing complexity and cost of fusion experiments necessitate international collaboration and data sharing. By adhering to FAIR principles, the fusion community can facilitate this broader engagement and cross-disciplinary utilization of valuable experimental data.

For the MDSplusML project, FAIR translates to:

- **Findable:** Implementation of a cross-machine data dictionary, potentially based on IMAS. This standardized dictionary will allow researchers from various institutions and disciplines to easily locate relevant data, even if they are unfamiliar with the specific experimental setup.
- **Accessible:** Development of a unified API for data access. This ensures that data can be retrieved through standardized methods, reducing barriers for external researchers or those working remotely. It also facilitates the integration of fusion data into broader scientific data repositories and platforms.
- **Interoperable:** Utilization of the IMAS dictionary to ensure data can be easily integrated and used across different systems. This interoperability is crucial for researchers working across multiple fusion devices or for those seeking to combine fusion data with information from other scientific domains.
- **Reusable:** Providing well-described data to enable diverse communities of consumers to utilize the information effectively. This includes comprehensive metadata, clear documentation of data collection methods, and standardized formats that allow for easy integration with various analysis tools and workflows.

By embracing these FAIR principles, the MDSplusML project not only enhances the utility of fusion data within the immediate research community but

also opens up new possibilities for collaboration and innovation. It allows for more efficient use of research resources, enables reproducibility of results across different institutions, and potentially accelerates the pace of discovery by allowing researchers from various backgrounds to bring new perspectives to fusion data analysis.

4. Proposed Approach

To address the challenges of providing efficient data access for ML pipelines, we propose a multi-faceted approach that combines data caching, parallelization, and API enhancements.

At the core of the strategy is the implementation of a robust caching system. The system will store popular quantities from all or most shots, as well as pre-computed data expressions, significantly reducing computation time for frequently accessed data. The cache implementation is designed to be flexible, allowing for variations between different platforms to optimize performance in diverse environments. The cache will operate in a 'data warehouse' mode. There will be regular jobs which update existing cached values to reflect new data analysis, and expand the list by caching the values of new cache misses. However, to ensure data integrity and handle unique requests, the system will maintain the ability to fall back on local native data access when necessary.

Alongside the caching system, we are developing new API calls to support efficient data retrieval. We have determined that the number of network transactions and network latency dominate data retrieval performance. Encouraging users to use MDSplus thin client protocol using `MDSplus.Connection.get()` to servers with fast access to data, both cached and uncached, is much more performant than the other MDSplus remote data access protocols. This is further improved if users group their data requests using `MDSplus.Connection.getMany()`. We have added a further enhancement **MDSplus.Connection.getManyMany(),** that allows the user to request a set of quantities from a list of shots. These functions are designed to minimize network transactions and optimize data retrieval for machine learning workflows that often require data from numerous experimental runs.

To validate our approach, we have tested various caching mechanisms. These tests used queries culled from disruption-py[7] for approximately 60 quantities and their time bases typically used in disruption-warning ML workflows. Our results show that performance varies significantly depending on the deployment environment. Before embarking on the implementation of caches, we did an assessment of the existing native MDSplus tools. This yielded some significant results. Small changes to user data access methods yielded significant performance improvements.

MDSplus includes several different network data access protocols which can be divided into two main classes. The distributed-client and thick-client protocols rely on the user's computer to evaluate expressions of the data. Clients request that the server execute low level I/O operations, returning binary buffers that are interpreted locally. While this offloads computation from the server, it results in very large numbers of network transactions per queried value. Data access for Alcator C-Mod data at the MIT Plasma Science and Fusion Center, is done this way by default. Over time the tradeoffs between compute power, both client and server, network latency, and network bandwidth have evolved. This had made the use of these protocols a particularly bad choice.

The MDSplus thin-client, and the new mdsthin[8] interface rely on the server to do all of the I/O and computation. User programs send expressions to be evaluated, often just MDSplus node references, to the server. The server computes the desired quantities and returns the answers over the network. This reduces the number of network transactions per query from potentially hundreds to less than ten. Assuming the server has sufficient capacity to service all of the concurrent requests, and does not itself need to rely on the network for I/O, this greatly improves performance. Figure 1 shows the time to retrieve 70 quantities and their timebases from 500 discharges using distributed-client and thin-client. This totaled ~1.2 MB of signal values and times from each shot. The tests were run on a set of two computers, one acting as the server, and the other as the client. The specification of the client had negligible impact on the performance, a virtual machine was used. The server is a 32 core older Intel server with direct attached spinning SAS storage. The network was the shared PSFC 10G backbone, with .3ms latency between the systems. As the number of parallel processes increases, bottlenecks on either the server or client limit the advantage of parallelism.

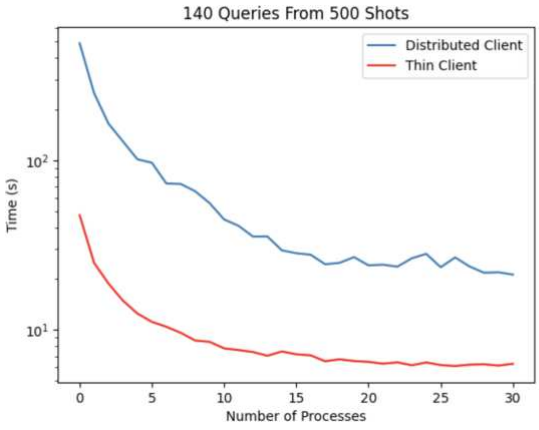

Fig. 1 Comparing distributed-client and thin-client data access time.

Care must be taken that the server has direct access to the underlying storage. If it in turn uses distributed-client to fulfill user data requests, no advantage is realized. Furthermore, if the underlying storage is provided to the server using high level protocols such as NFS or CIFS, much of the performance gain is lost.

MDSplus uses a search list to specify the location of the files that hold the trees. Each term of the list can optionally start with a '`hostname::`' and is followed with a file specification (on that host) with some parts

encoded as `'~x'` where `x` can be `a` `a-j'` which are digits of the shot number or `'t'` for the name of the tree. For example for shot 1090909009 and treename cmod, the tree search clause specification `alcdata-archives::/cmod/trees/archives/~i~h/~g~f/~e~d/~t` corresponds to a network connection to the computer `'alcdata-archives` and the directory specification `'/cmod/trees/archives/09/09/09/cmod/'`. The tree path used by client computers at the PSFC is:

```
echo $default_tree_path
alcdata-
test::/cmod/trees/test/~t/;
alcdata-
new::/cmod/trees/new/~t/;
alcdata-
models::/cmod/trees/models/~t/;
alcdata-
archives::/cmod/trees/archives/~
i~h/~g~f/~e~d/~t;
alcdata-
saved::/cmod/trees/saved/~t/
```

When a user attempts to open a tree, each of these places is queried in turn until the files are found or the list is exhausted. The default data server which houses the test trees, new shots, and models locally, uses distributed I/O to access the archived data.

```
echo $default_tree_path
/cmod/trees/test/~t/;
/cmod/trees/new/~t/;
/cmod/trees/models/~t/;
alcdata-
archives::/cmod/trees/archives/~
i~h/~g~f/~e~d/~t;
/cmod/trees/saved/~t/
```

This is quite good for test and new shot access, but incurs a high cost for archived data. We can further improve matters by connecting directly to the archive server and telling it that the files are either present in the archive or do not exist. In Figure 2 the thin-client and shortened-path traces show this improvement.

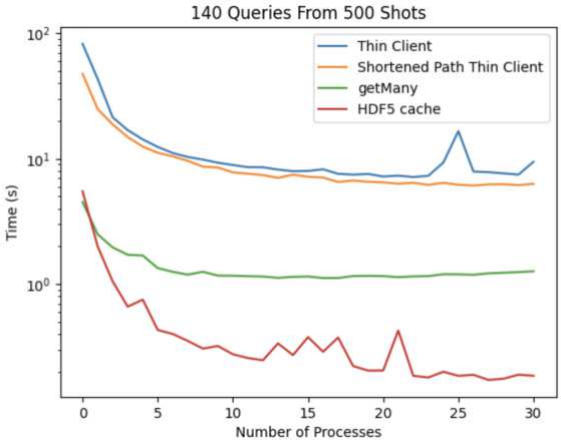

Fig. 2 Effect of shortening path, getMany, and caching on data access time.

We can further improve performance by grouping data requests. As seen in the 'getMany' trace in Figure 2. In this mode, the user builds a list of requested signals and sends it over the network in one request. The server fulfills the request, and sends all of the answers back in one transaction.

There is further performance improvement when the requested values are pre-computed, stored in a cache and read directly by the client. The HDF5 cache trace in Figure 2 shows this. For on premise computation, the gains are not sufficient to justify the effort. However, in cloud environments cached values provide several advantages.

Native data access from the cloud may not be possible or desired. Local firewalls, data security policies, software installation, and availability of support libraries may preclude it. Furthermore, high network latencies, some of which may be partially mitigated by the techniques described above, make this unattractive. A cache of precomputed, well described, vetted quantities allows faster and more FAIR data access.

4.1 Data Caching

For environments where local MDSplus access is not feasible, the data server behind MDSplus thin-client can instead return data from pre-populated caches. Depending on the policy for cache misses, data access for collaborators in this environment can be controlled. The data owners can 'publish' quantities from a set of shots for collaborators' use. We can optionally allow native data access for cache misses for authorized users. However, there are performance implications of trying to do native access over the wide area network, so this choice, to try native access or not, should be taken carefully. This should be relatively easy to implement. The cached quantities can be marked as missing, and queries for them can be quickly failed. If, on the other hand, there is no entry then policy can dictate the native access attempt behavior. In most cases with pre-populated caches, a cache miss means that the data did not exist when the cache was built, therefore falling back to local access is both ineffective and inefficient.

Various caching mechanisms have been tested and benchmarked using the same queries as above (approximately 70 quantities from 10,000 shots) that are typically used in disruption-warning ML workflows. The performance of different caching schemes varies significantly depending on the deployment environment. As described above, on premise applications can get good performance from properly configured native access methods. For cloud-based applications, and to accommodate wider groups of data consumers constructing caches of precomputed, well vetted quantities is worthwhile.

We have tested several caching technologies using the inputs to the data preparation code disruption-py as a representative collection of user queries. Disruption-py computes quantities relevant to machine learning tasks related to disruption prediction from approximately 70

raw and processed quantities and their time bases for each shot.

HSDS is an API that conforms to the HDF5[9] python interface 'h5py'. It is a RESTFUL[10] service that returns HDF5 datasets on demand. Scalability is achieved by a two-layer architecture which can be run under Kubernetes to provide scalability. Initial tests however were disappointing because, as found with the native MDSplus data interfaces, the performance is inversely proportional to the number of network transactions. Each data request is a separate transaction from client to the server, and then the server in turn makes network transactions to fulfill the request. We could gather the requests in a similar manner to **`getMany`** described above to mitigate this. However, the standard HSDS configuration did not provide a good match for our use case.

Mongodb is an in-memory database which can have very good performance. It suffers from similar performance issues unless the IO requests are aggregated. If they are, then very high performance can be achieved.

Native HDF files can provide very high performance. Figure 2 shows the achievable data retrieval rates. On Amazon Web Services S3 storage provides very high throughput data access, but single stream queries are quite slow. To obtain good performance, we must either use a very large number of threads, or some parallel prefetch of the needed files.

We are experimenting with a variation on HSDS called H5Image to improve the performance of the HSDS based queries. Figure 3 shows the architecture.

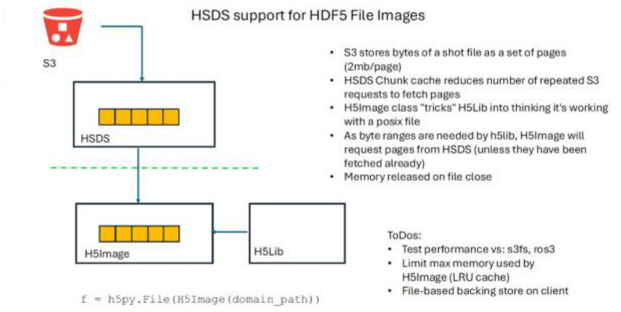

Fig. 3 HSDS support for HDF5 File Images.

The optimization of data access performance in fusion research hinges on several key factors. In today's technological landscape, bandwidth is generally plentiful and relatively affordable, which is advantageous for data-intensive operations. However, latency remains a significant challenge, often proving to be a stubborn bottleneck that resists easy solutions. This is particularly relevant in fusion data access, where the typical patterns of data retrieval are such that transaction costs can quickly negate any benefits gained from high bandwidth.

The shift towards cloud-based storage solutions presents a mixed bag of opportunities and challenges. While these platforms often boast impressive throughput capabilities, they can also come with hefty transaction costs, which can be a significant drawback for certain types of data access common in fusion research. To navigate these challenges and achieve optimal performance, especially in cloud environments, it's becoming increasingly necessary to implement parallel request strategies. This approach allows for more efficient utilization of available resources and can help mitigate some of the latency and transaction cost issues inherent in accessing large volumes of fusion data.

5. Future Work and Conclusions

Based on the findings from initial testing and performance analysis, the following plan has been outlined for further development:

5.1 API Development

Implement a getManyMany API that allows users to request a list of quantities from multiple shots. A prototype of this has been written. This provides a significant performance improvement over MDSplus thin-client and the single shot getMany APIs. Existing user programs will require changes in order to take advantage of these APIs. Programs written using the MDSplus object interface must be converted to use the functional Connection.get(). They can get further performance improvement by using Connection.getMany(), or Connection.getManyMany().

Shot to shot parallelization in the prototype getManyMany is done in the client program. We are considering delegating this to the server, which would trade complexity for flexibility on how the task is broken up.

5.2 Data Warehouse and Caching

A generalized cache of pre-computed values will be developed and maintained. Users who are not on premise will be able to use this to get fast access to vetted well described data. The IMAS data dictionary with possible extensions will be used for names and definitions in at least part of this cache. The cache will be updated on a regular basis to reflect new data requests, and updates to the referenced quantities. The scheme will be agnostic to the underlying caching technology. We will continue to refine and optimize caching strategies based on ongoing performance testing.

6. Conclusion

In conclusion, the MDSplusML project aims to bridge the gap between traditional fusion data management and the requirements of modern machine learning pipelines. By implementing intelligent caching, optimized APIs, and adhering to FAIR principles, the project seeks to provide researchers with fast, efficient access to the large-scale datasets required for ML applications in fusion research. Ongoing work will focus on refining these approaches and testing them in various deployment scenarios to ensure optimal performance across different research environments.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work the author(s) used [Claude 3.5 Sonnet] in order to improve the readability of the text. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

References

[1] Stillerman, J. A., et al. "MDSplus data acquisition system." Review of Scientific Instruments 68.1 (1997): 939-942.

[2] "MDSplus data system," MIT Plasma Science and Fusion Center, April 2024, https://mdsplus.org/index.php/Introduction

[3] "Highly Scalable Data Service (HSDS)", The HDF Group, https://www.hdfgroup.org/solutions/highly-scalable-data-service-hsds/, referenced August 2024.

[4] "MongoDB", MongoDB, Inc., April 2024, https://www.mongodb.com/, referenced August 2024.

[5] Wilkinson, M., Dumontier, M., Aalbersberg, I. et al. The FAIR Guiding Principles for scientific data management and stewardship. Sci Data 3, 160018 (2016). https://doi.org/10.1038/sdata.2016.18

[6] Imbeaux, F., Pinches, S.D., Lister, J.B. et al. Design and first applications of the ITER integrated modelling & analysis suite. Nucl. Fusion 55, 123006 (2015). https://doi.org/10.1088/0029-5515/55/12/123006

[7] Rea, Cristina, Trevisan, Gregorio Luigi, MIT PSFC Disruption Studies Group, DisruptionPy: An open-source physics-based Scientific Framework for Disruption Analysis of Fusion Plasmas, https://doi.org/10.5281/zenodo.13935223

[8] "mdsthin python package", https://pypi.org/project/mdsthin/, referenced August 2024.

[9] https://www.hdfgroup.org/HDF5/, referenced August 2024.

[10] Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine. Available at: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Acknowledgements

This work is supported also by the U.S. Department of Energy, Office of Science, Office of Fusion Energy Sciences, under Award DE-SC0024368