A Dynamic, Multi-Protocol Data Storage Integration Framework for Multi-Tenanted Science Gateway Middleware

1st Dimuthu Wannipurage
Cyberinfrastructure Integration Research Center
Indiana University
Bloomington, USA
dwannipu@iu.edu

2nd Suresh Marru
Cyberinfrastructure Integration Research Center
Indiana University
Bloomington, USA
smarru@iu.edu

3rd Eroma Abeysinghe
Cyberinfrastructure Integration Research Center
Indiana University
Bloomington, USA
eabeysin@iu.edu

4th Isuru Ranawaka
Cyberinfrastructure Integration Research Center
Indiana University
Bloomington, USA
isjarana@iu.edu

5th Marcus Christie
Cyberinfrastructure Integration Research Center
Indiana University
Bloomington, USA
machrist@iu.edu

6th Marlon Pierce
Cyberinfrastructure Integration Research Center
Indiana University
Bloomington, USA
marpierc@iu.edu

***Abstract*—Science gateways are user-centered, end-to-end cyberinfrastructure for managing scientific data and executions of computational software on distributed high performance computing and cloud resources. A typical goal of a science gateway is to help users make more effective use of high performance and other research computing resources, thus easing the user support burden on research computing centers while also increasing their ability to scalably serve user communities. An important challenge for science gateways is to manage data at scale, where scaling pressures come from both the number of users of successful gateways and the size of data used in scientific workflows that the gateways execute. This paper examines the use of managed file transfer (MFT) approaches to generalize several data flow scenarios. We examine a particular implementation, Airavata MFT, which can be used to extend the data stores integrated with a science gateway beyond local storage to include multiple remote storage instances, including cloud vendors. An agent-based approach is adopted, which enables streamlined end-to-end flows of data between user storage and backend computing. Integration with these diverse storage types is done through a common application programming interface for data operations, making the integration of a new storage system a configuration change without requiring changes to portal server code.**

***Index Terms*—Science Gateways, Managed File Transfers, Distributed Storage, Apache Airavata**

I. Introduction

Science Gateways [1] play a vital role in research cyberinfrastructure by bridging the gap between scientists, research computing, and data management infrastructure. They provide domain-specific user environments while absorbing complexities such as using job schedulers, storage infrastructure, and scientific software, and they further can absorb local policies and security compliance issues, such as temporary/scratch space usage, job queue limits, and proper resource usage. They thus can simplify the use of scientific computing resources, reducing the support burden on research computing support operations.

Mature software projects [2]–[6] exist to create science gateways. Science gateway platforms further simplify the process of creating and operating science gateways by offering hosted deployments, which may be multi-instance or multi-tenanted [7].

This paper investigates three current research problems data management for science gateway systems: 1) decoupling data movement from web and control traffic in science gateway cyberinfrastructure; 2) interfacing with heterogeneous storage systems that include POSIX file systems [8], Object Stores [9], and Block Stores [10]; and 3) supporting new transfer protocols.

II. Data Management Challenges for Science Gateways

Data Movement in Multi-Tiered Architectures: Science gateway cyberinfrastructure is typically conceptualized using a multi-tiered architecture [11], as illustrated in Fig. 1. A web server hosts the gateway portal that interfaces with the user's

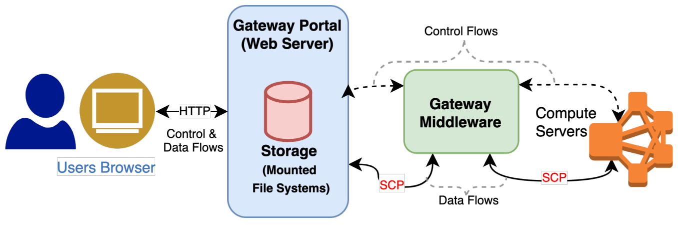

Fig. 1. A conceptual view of canonical gateway architecture with four distinct layers. Data moves between users' local environments and remote computational clusters or storage devices through multiple hops of tiered services. These extra hops constrain data management flexibility, impede the design of highly available gateway systems, and add latencies and unnecessary network traffic.

browser. The second tier bundles gateway middleware services such as API servers, state management and orchestration services, identity and security services, and persistent databases. The third tier consists of remote computing and data storage systems. The general challenge is the movement of data and management of permissions across tiers, where the browser (and its host computer), the portal server, the middleware, and the backend services are controlled by different groups.

In a canonical deployment (Fig. 1), transferring users' data to and from computational and storage resources is typically done by first copying to a large file system mounted to the gateway portal's web server. We will refer to this storage as the "gateway portal data store." Having the data locally deployed with the gateway portal web server enables the portal to enforce data security with user-level authorization and serve the data to users through standard HTTP uploads and downloads.

In a subsequent step, such as using the data as an input to an analysis or simulation, the middleware pulls the file from the gateway portal data store using standard protocols such as SCP and pushes the data stream to compute resources using SCP, SFTP, or similar protocols. The middleware may use in-memory buffering to optimize these transfers. This approach is rudimentary but provides a simple approach to developing end-to-end data transfer solutions. However, the extra and unneeded hop of transfers is inefficient and limits scalability of users and data.

Science gateways execute scientific workflows and processing pipelines, so data are inputs and outputs of these processes. Gateways are responsible for fetching input data from users, submitting computational jobs to remote resources, and managing the outputs. Traditionally, users upload input data through gateway portals to the gateway portal data storage to make it available for the gateway middleware so that the middleware can use these inputs to submit jobs into remote computational resources.

One of the main assumptions made in this scenario is that the gateway portal data storage and the portal server are deployed on the same server so that the portal can upload and download data by utilizing the local file system calls on top of the storage directory. Even though this is simple and easy to implement, this approach sacrifices the scalability of gateway storage as it is bounded by the maximum disk space available in the host machine or any directly attached network storage.

Another gateway use case is the capability of configuring and using cloud data storages as gateway storage. Cloud storage systems such as Box and Google Drive provide inexpensive, easy to use, and significantly high volume storage solutions for academic institutions, and as a result, researchers often store their research data in such cloud endpoints. When it comes to processing such data through a science gateway, it is cumbersome and inefficient to download and reupload that data into the gateway storage. A better solution is to provide direct access from the gateway to these cloud endpoints rather than the typical POSIX-based storage acting as the broker.

We propose to address the above challenges by providing more flexibility in how science gateway cyberinfrastructure (Fig. 2) can mount storage endpoints and manage direct data transfer paths between storage endpoints and compute resources, eliminating data routing through the gateway middleware, which retains coordination functions.

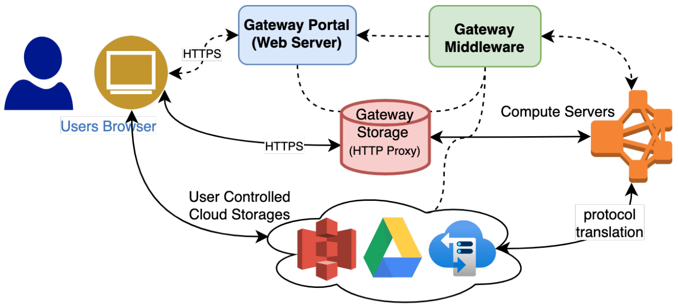
Gateway Portal (Web Server)
Gateway Middleware
HTTPS
Users Browser
HTTPS
Gateway Storage (HTTP Proxy)
Compute Servers
User Controlled Cloud Storages
protocol translation

Fig. 2. Decoupling the gateway's storage from the webserver and serving by an HTTPS proxy eliminates redundant transfer hops while providing flexibility to add external storage without necessarily limiting web server deployments, high availability, and uptimes. The decoupled architecture also provides the capability to add user-controlled cloud storage but requires protocol translation and secured credential delegation. The dotted lines in the figure indicate control flow and solid lines data flow.

III. An Extendable Managed File Transfer Framework for Science Gateways

To support the growing number of data transfer use cases, we advocate the distributed form of the "separation of concerns" design approach, collecting the data management and transfer functionality into a separate component, or service, to avoid introducing too much complexity into either the gateway portal server or the middleware. This approach has the added potential benefit that the service, if properly designed, should work in standalone usage scenarios as well as in part of a fully integrated science gateway platform such as Apache Airavata [6]. Airavata MFT is a novel, extendable and multi-protocol managed file transfer framework that is designed to address the above mentioned aspects.

Airavata MFT has two main components: MFT Back-End Services and MFT Agents (Fig. 3). MFT Back-End Services are a set of microservices that work together to accept, monitor, and retry a transfer request coming into the framework.

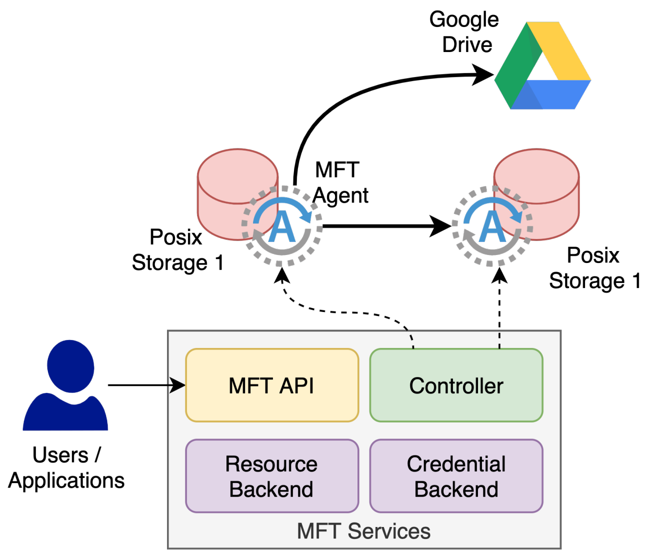

Fig. 3. Top level Airavata MFT with two main data transfer modes; agent-to-agent transfer and agent-to-storage transfer. Thick solid arrows depict data transfer paths and dashed arrows depict transfer control paths.

These are middleware services that would typically be co-deployed with other Apache Airavata components but which may also be deployed as standalone services. These services comprise a set of back-end components that control the data management processes requested by clients. These back-end components can be categorized into four sections: MFT API, Controller, Resource Back-End, and Secrets Management Back-End. MFT API is the front facing interface that accepts data management and transfer request from users. Once the requests are validated, the API component hands the request over to the Controller to coordinate the execution of the request. Airavata MFT supports a wide variety of storage types to be integrated. The Resource component registers and keeps track of these storage types, and data reside inside these storage systems. Finally, the Secrets Management component supports the the secure storage and access of resource credentials needed to connect to the variety of resources in the Resource component.

MFT Agents are the daemons installed in external storage systems that perform actual data transfers based on the instruction received by the MFT back-end services. MFT Agents are installable binaries bundled with library implementations for protocols like SCP, SFTP, FTP, HTTP, and TUS. For cloud vendor systems such as Amazon S3, Google Drive, Azure Blobs, Box, and DropBox, these protocols and associated security mechanisms can be embedded as client libraries. These agents can be directly installed on target platforms. For example, an agent can be deployed on the same host as the storage or on a network level closer to the storage endpoint. These agents are capable of communicating with any storage endpoint type using supported protocols to either push or pull data using bundled protocol libraries.

There are two modes of data transfer types in this approach: agent-to-agent and agent-to-storage transfers. In the agent-to-agent mechanism, the sending agent pulls file content from its local file system and pushes to the agent installed on the target storage. Agent-to-storage transfers have an agent on one end that communicates with the other storage endpoint using the protocol that the target storage endpoint supports. That is, an agenty may directly communicate with a remote storage endpoint using SFTP, for example.

Airavata MFT is designed to separate data transfer paths and control paths at the architectural level. This provides flexible control of the data transfer route and allows for the implementation of end-to-end transfer qualities of service, such as encryption and integrity verification. Fig 3 shows the command and transfer path separation of a general data transfer job handled by Airavata MFT. Users send data transfer requests to the MFT API, which forwards that message to the Controller. The Controller determines the corresponding agents that must be notified regarding the transfer. Resource and Credential Back-End components provide the connection details and credentials for those agents to connect securely to the other endpoints. Once those control messages are sent to the agent, the agent performs the transfer in a separate channel.

IV. Integrating Airavata MFT into Science Gateways

A traditional Science Gateway might handle data at multiple stages. First, users upload input data for HPC jobs using the gateway portal into a gateway data storage. Then those data are copied to the HPC cluster to execute the job. Once the job is completed, output data is copied back to the gateway storage for users to download. For each of these steps, Airvata MFT can offload most of the data transfer and management functionalities from the gateway middleware and potentially perform those task in a more efficient and scalable manner.

A. Managing, Uploading and Downloading Data through the Gateway Portal

In a traditional deployment, a science gateway's portal server comes with limited local storage attached that is used to store user-uploaded input data and job output data; gateways typically expect users to download and manage their output data and may implement periodic archiving or removal. In this case, data management consists of file operations such as locally copying data, creating directories, listing files and removing data. Fig. 4 shows how a gateway portal translates users' data management, upload, and download requests using local Linux file system commands. However, as mentioned earlier, this approach is not scalable because if the local storage is exhausted, the only ways to gain more capacity are to mount more storage into the same instance, or to remove or archive older data sets or data sets above a certain size threshold.

Using Airavata MFT, we can logically and physically decouple science gateway portals from their local storage so that those can exist in two different instances. All the local file systems calls that were executed by the portal can be replaced from Airavat MFT file system API calls in order to minimize the amount of changes going into the portal source code.

B. Utilizing Airavata MFT for Non-Intensive File Operations in a Gateway Portal

Fig. 5 shows how the proposed architecture with Airavata MFT integrates with a gateway portal for non-intensive file management operations. Compared to earlier local data storage approach, here we can have one or many external data storage servers. MFT Agents acts as the interfacing mechanism between the portal and the storage endpoint. In this case, Agents can be installed inside the storage server (as in External Data Storage 2) or closer to the storage server's network (as in External Data Storage 1). Depending on the locations where those are placed, Agents can change the communication mechanism that they need to interact with the storage endpoint. For example, if the Agent was placed inside the storage server, it can communicate with the storage with local file system calls. On the other hand, if the Agent is remote to the portal server, it can communicate with a standard file management protocol like SFTP. In both cases, the MFT Agent provides a single set of API methods that implement these operations. The MFT API is specified using gRPC, which generates client bindings in multiple programming languages.

MFT Services communicate with MFT Agents through a secure Agent communication protocol to pass over the data management operation requests received from the gateway portal. Selection of the correct MFT Agent to communicate is performed inside the controller back-end service considering the locality of the Agent and the target storage endpoint.

C. Utilizing Airavata MFT for Data-Intensive File Operations in a Gateway Portal

In a gateway portal, data intensive operations include job input file uploads and output file downloads once jobs are completed. As the content of the data file can vary from few bytes to several gigabytes or more, we cannot use the gRPC API provided by MFT Services to stream the file content to the user's browser as gRPC is not optimized for large data transfers. In the classical local file storage approach, this is not an issue because the portal can locally read the file content and provide the byte stream through Hypertext Transfer Protocol (HTTP) into user's browser. At the same time, users are using web browsers to upload and download data so HTTP is the only viable protocol to use as the medium to transfer data at the front end.

We call theses as synchronous data transfer paths where the user is actively waiting until the data transfer job is completed. To address this, any MFT Agent deployed in an Airavata MFT deployment comes with a public HTTP endpoint exposed to the outside. In an synchronous data transfer scenario, users can use this public HTTP endpoint to upload or download data content. However, as Agents are managed and registered at the MFT back-end service level, users do not have direct knowledge on which agent to connect and target path to upload or download data. Because of that, we introduce an initial

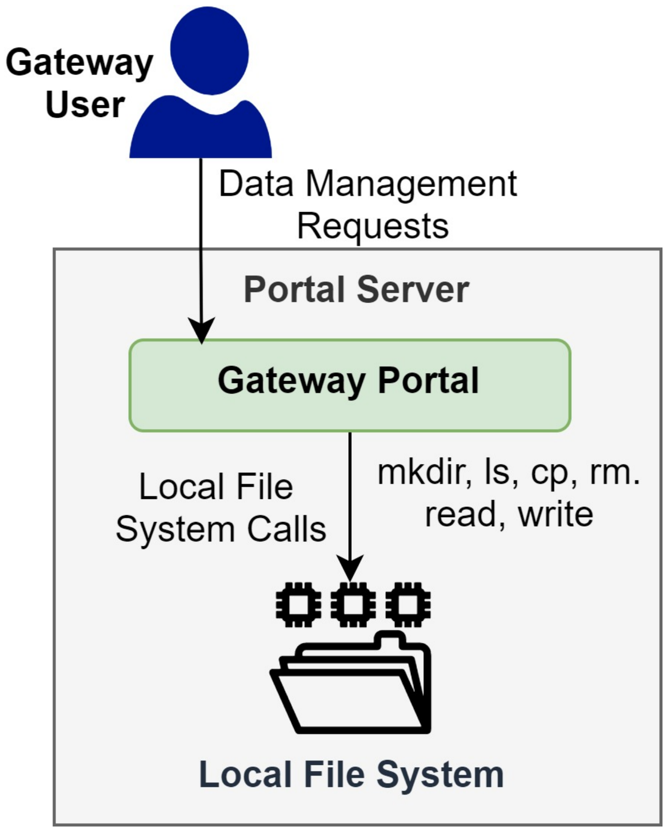

Fig. 4. Data management in a classical gateway portal is performed through utilizing local file management APIs provided by the operating system. Because of that, portal and the storage file system should reside the same server

control path that involves gateway portal and controller back-end to figure out the target agent URL and data path to upload and download data.

Fig. 6 shows full control and data paths for a synchronous data download scenario using Airavata MFT. The number placed at each arrow represents the order of execution and operations performed at each stage are described as below

1) User (or Web Browser) sends data download request with the resource path and endpoint name to the gateway portal.
2) Gateway portal forwards the download request to MFT Back-end services using the gRPC API.
3) The Controller back-end of MFT Services accepts the request and determines the most suitable Agent to handle to request by calculating the locality of the storage and registered Agent pool. Once an Agent is selected, controller sends a download request to the Agent through Agent's communication channel.
4) Agent processes the message and provides a unique download URL, which consists of the <Agent's public endpoint URL>/<unique id>to the controller as the response. In addition, the Agent keeps a copy of the URL with download request inside its request cache.
5) Controller sends new download URL to the gateway

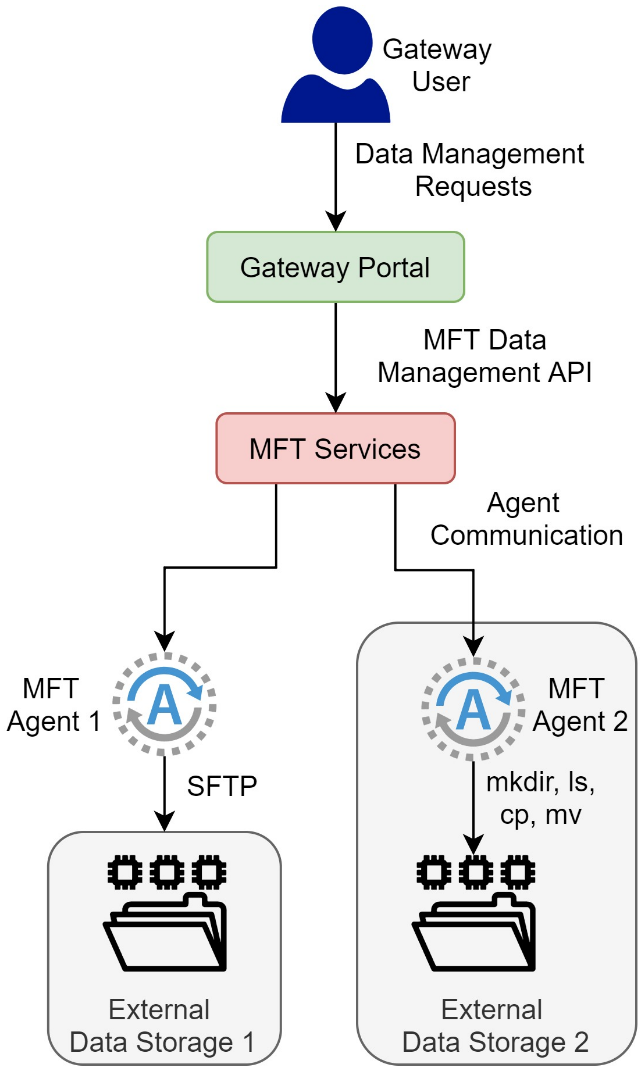

Fig. 5. Using Airavata MFT to connect and manage external data storages into the gateway portal by replacing local file system calls with MFT Data Management API. MFT Agents can live inside or closer to the storage endpoints.

portal as the response for step 2.

6) Gateway portal returns the same URL as a HTTP redirect response to the user (web browser) as the response for step 1.
7) User (with web browser) visits the new URL, which invokes the HTTP request handler of the target Agent.
8) Agent verifies the request and incoming user against its message cache.
9) If the request is a valid one, Agent starts to fetch data from the endpoint using a supported protocol (SFTP or local file system calls) and translates into a HTTP data stream as the response from the user.

D. Integrating Cloud Storage Endpoints

So far we have considered how an MFT service can be used to expand a gateway's data storage to include separate server instances that can be accessed either through an Agent's

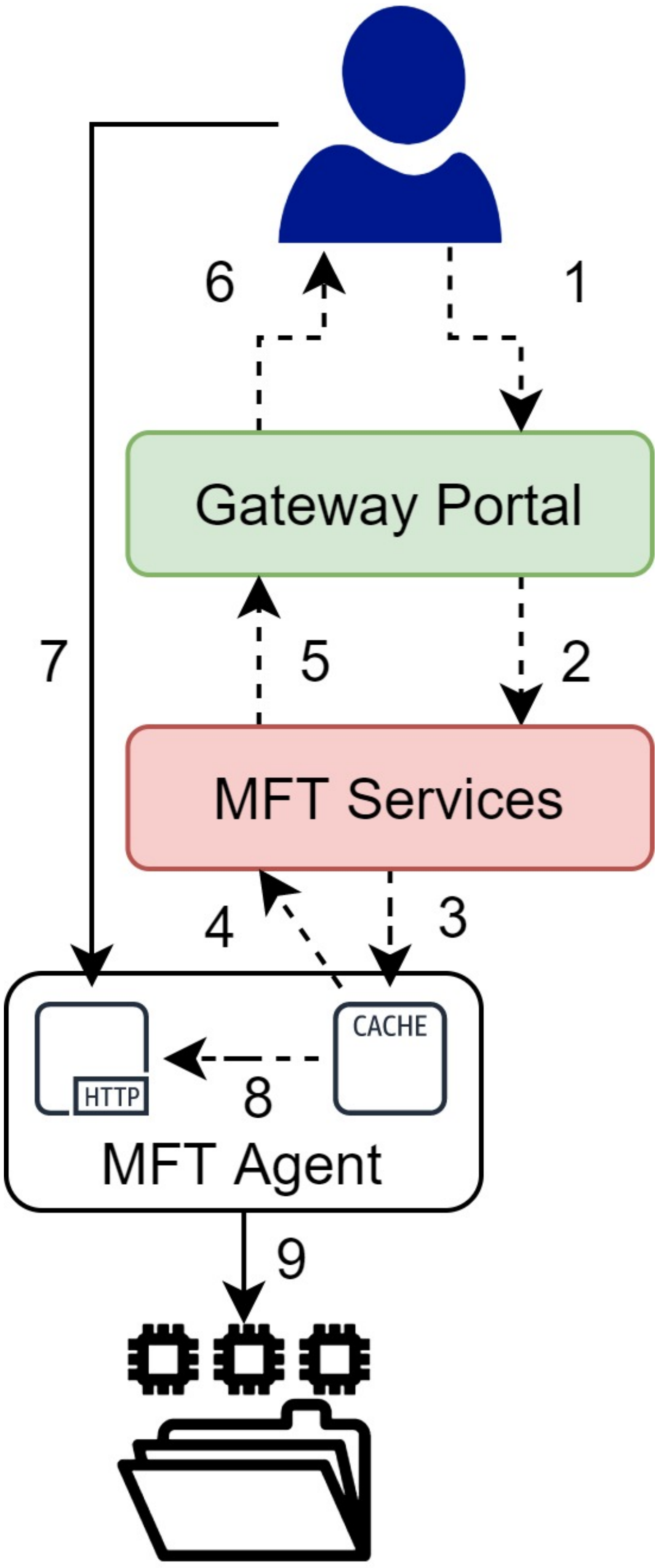

Fig. 6. Control and data paths for active data transfers between gateway portals and remote storage endpoints using Airavata MFT. Solid arrows represents data paths and dotter arrows represent control paths

local file system interface or the SFTP protocol. However, the basic approach can be generalized to include MFT Agents that connect to commercial cloud storage endpoint types as well. The current distribution of the MFT Agent is bundled with Amazon S3, Google Drive, Azure Blobs, Box, and DropBox cloud storage adaptors. A data translation engine inside the Agent has the capability to read from any supported storage type and convert the data stream to the destination storage type at the streaming level (Fig. 7). In addition, the layers above the MFT Agent including the MFT Services and gRPC APIs use a generic file management API. This enables a gateway portal, once it is using the MFT API, to transition between local, remote, and cloud storage systems through configuration changes rather than code changes.

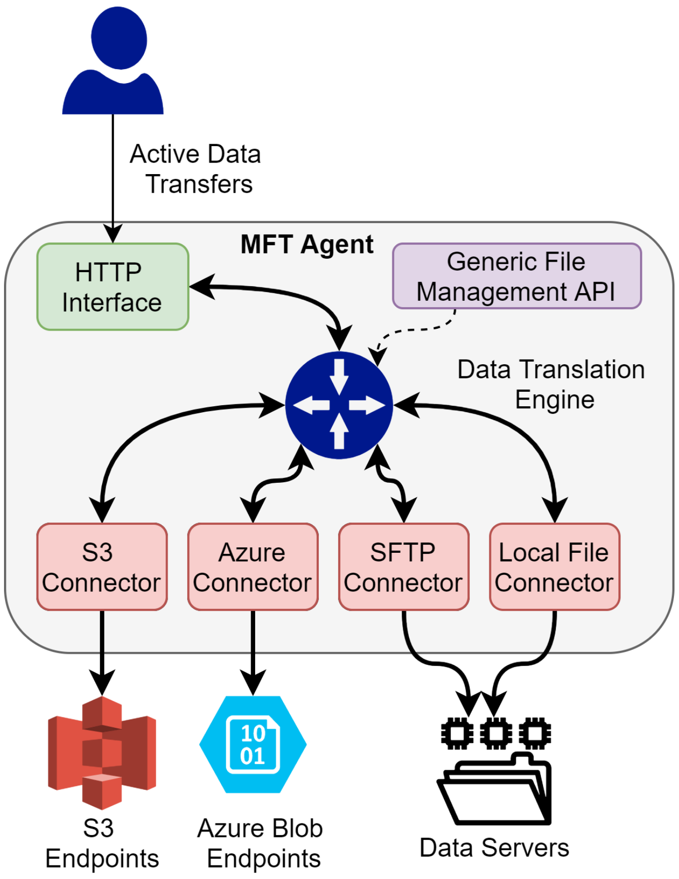

Fig. 7. In depth view inside the MFT Agent on how it handles cross protocol transfers. An Agent comes with built-in protocol connectors, and each connector is wired with the data translation engine. These connectors implement the generic file management API in a provider- or vendor-specific manner. Dotted arrows represent control paths and solid arrows represent data paths.

E. Integrating User-Provided Storage

With the popularity of cloud storage solutions, it is important that we consider how user-provided storage can be integrated seamlessly with the gateway portal server. User-provided storage may be both a source for input files and a destination for outputs generated by the gateway. This integration can also alleviate the storage requirements users place on gateways: user-provided storage can host files that are too large or (in advanced cases) have additional security considerations.

As discussed previously, MFT Agents provide generic data management APIs that can be implemented by the gateway portal server. This is a variation on the previous scenario, in which the gateway provider provisions cloud storage. In the current case, the user-provided storage works in a similar fashion, but with additional access controls enforcing read and write permissions.

F. Data Transfers Between POSIX Storage and HPC clusters

In a typical science gateway workflow, once the input data are deposited in to the gateways storage, the next step is to transfer the data to the HPC cluster as inputs for computational jobs. Airavata MFT separates the middleware, which controls the transfer, from the data transfer path, which is handled by agents; see Fig. 8. If both endpoints have agents installed, then those agents open a HTTP2-based TUS [12] data channel to transfer data in a secure and resumable manner. If only one endpoint has an agent installed, that agent uses the supported protocol on the other endpoint to transfer data.

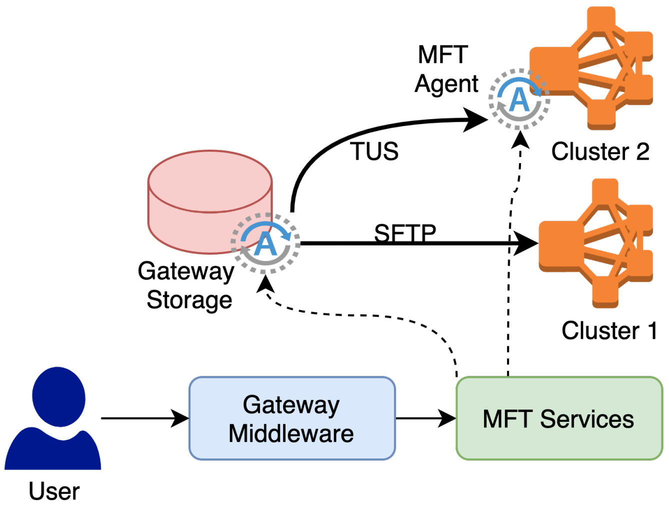

Fig. 8. Implementing data transfer paths from gateway storage to the HPC cluster with Airavata MFT. Thick solid arrows depict data transfer paths and dashed arrows depict transfer control paths.

G. Data Transfers Between Cloud Storage and HPC Clusters

When integrating cloud data storage, we can either install MFT Agents near the cloud storage or use an existing agent installed on a cluster to pull or push data using a cloud storage supported protocol (Fig. 9). When installing a cloud MFT agent, we should make sure that the agent is installed in a place inside the same network subnet of the cloud provider. For example, S3 cloud agents can be installed within an AWS virtual machine. This approach provides the maximum transfer throughput with added network-level security between the cloud storage and the MFT Agent.

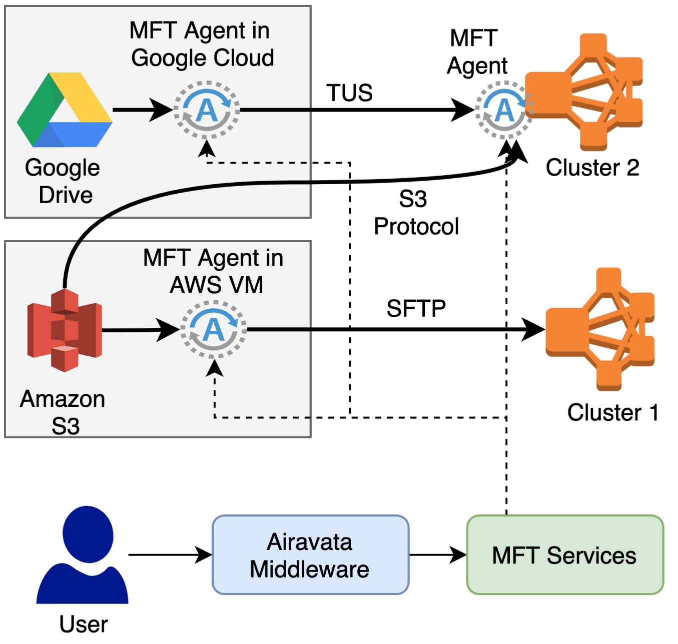

Fig. 9. Handling cloud storage integration into gateways using Airavata MFT. Solid arrows depict data transfer paths and dashed arrows depict transfer control paths.

V. Internal Communication Security

Airavata MFT is a collection of microservices working together and it is vital to have each communication section is secured so that no one from outside can see the data flowing through in any channel. Fig. 10 shows all messaging paths and underlying messaging protocols among Airavata MFT microservices and external entities. Communications between back-end components are all performed through the gRPC protocol. Controller to Agent communications are performed through a Consul asynchronous messaging medium and Consul connections are handled through HTTPS protocol. MFT Agents are getting external connections from users for active data uploads and downloads through HTTPS protocol. gRPC server sockets in each service is configured with SSL/TLS encryption and clients connections are made through a secure channel. Consul server is configured to expose its HTTP API through a TLS channel which makes sure that both connections from the controller and the Agents are secured and encrypted. HTTP endpoint of the MFT Agent is a Netty based non blocking web server configured with TLS so the connections coming from users to Agents are also encrypted. As summary, all the connections between MFT components and external entities are encrypted over SSL/TLS so no one can tamper with the data flowing through any channel.

However, encryption over communication channels alone do not guarantee a secure communication. Each message should be authenticated and authorized before preforming any action over it. To address this, we provide a dual level authorization mechanism for inter service communications. All messages that are coming into MFT Services are initiated by external

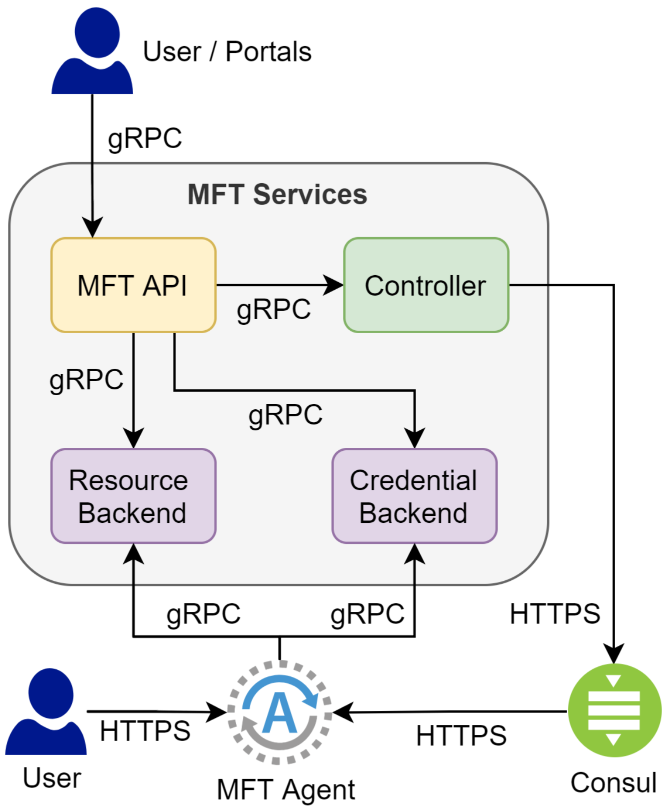

Fig. 10. Communication protocols between the components of Airavata MFT and rest of the external entities

users. We expect these users to be authenticated against an identity provider and pass an authorization token along with requests they sent into to MFT API. Each back-end service has a message interceptor to validate this authorization token before forwarding into the the message pressing handlers of the service. If the front facing service needs to connect to any internal service, this user token is forwarded with the request message. In addition to that, each service has a service account registered with the same identity provider mentioned above. If an internal service receives a message from a front facing service, it expects user token and the incoming service's service account token at the security interceptor to validate both user's and the sending service's authenticity and permissions. Fig. 11 shows how a user request is navigated to the controller back-end with dual level authorization mechanism.

VI. Related Work

Prior work related to the capabilities discussed in this paper can be separated into storage management and data movement. There are multiple related efforts that focus on subsets of these capabilities. For instance, storage solutions such as MinIO [13] and Ceph [14] provide a unified object storage abstraction mapping multiple file systems.

Globus [15] is well known for scientific data management, particularly in the transfer of very large data sets. Globus data transfer services utilize GridFTP, a high performance, point-to-point data transfer protocol. The main advantage of this

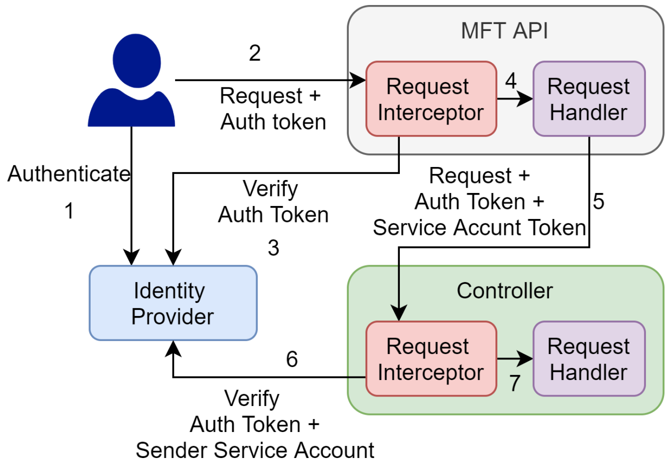

Fig. 11. Example dual level authorization between MFT services. Index at each step represents the order of execution

protocol is that the data and control paths are separated. This way, data can be transferred directly from one point to another while keeping the control path for another third party. Globus software is closed source, and its operations are proprietary.

Open source projects like StorkCloud [16] and Rclone [17] are MFT implementations that provide extensible multi-protocol transfer job schedulers and directory listing services. However, the data and control paths are interleaved in these approaches; as we have discussed here, separation of data and control paths is key for science gateway use cases. Alluxio [18] is a data orchestration framework that is heavily used in data analytics applications. Alluxio is a fully contained distributed system with its own data catalog and credential management system, making it challenging to integrate into other systems.

Security considerations for the hybrid middleware-distributed agent approach that we adopt here are described in more detail in [19].

VII. Conclusion and Future Work

In this paper we discussed our approach for diversifying data and storage management for science gateways. Motivated by typical data flow and execution scenarios common to many science gateways, the paper focuses on specific use cases that decouple data movement from both gateway web servers and middleware to enable direct transfers between users' systems and computational resources. The capabilities discussed in this paper lay the foundation for empowering gateway frameworks to support diverse data-intensive computing such as more ubiquitous integration of cloud storage systems and support of data distribution gateways for scientific instruments.

References

[1] K. A. Lawrence, M. Zentner, N. Wilkins-Diehr, J. A. Wernert, M. Pierce, S. Marru, and S. Michael, "Science gateways today and tomorrow: positive perspectives of nearly 5000 members of the research community," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4252–4268, 2015.

[2] V. Jalili, E. Afgan, Q. Gu, D. Clements, D. Blankenberg, J. Goecks, J. Taylor, and A. Nekrutenko, "The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2020 update," *Nucleic acids research*, vol. 48, no. W1, pp. W395–W402, 2020.

[3] M. McLennan and R. Kennell, "Hubzero: a platform for dissemination and collaboration in computational science and engineering," *Computing in Science & Engineering*, vol. 12, no. 2, pp. 48–53, 2010.

[4] D. Hudak, D. Johnson, A. Chalker, J. Nicklas, E. Franz, T. Dockendorf, and B. L. McMichael, "Open ondemand: a web-based client portal for hpc centers," *Journal of Open Source Software*, vol. 3, no. 25, p. 622, 2018.

[5] P. Calegari, M. Levrier, and P. Balczyński, "Web portals for high-performance computing: a survey," *ACM Transactions on the Web (TWEB)*, vol. 13, no. 1, pp. 1–36, 2019.

[6] S. Marru, L. Gunathilake, C. Herath, P. Tangchaisin, M. Pierce, C. Mattmann, R. Singh, T. Gunarathne, E. Chinthaka, R. Gardler *et al.*, "Apache airavata: a framework for distributed applications and computational workflows," in *Proceedings of the 2011 ACM workshop on Gateway computing environments*, 2011, pp. 21–28.

[7] M. E. Pierce and S. Marru, "Integrating science gateways with secure cloud computing resources: An examination of two deployment patterns and their requirements," in *2020 IEEE/ACM International Workshop on HPC User Support Tools (HUST) and Workshop on Programming and Performance Visualization Tools (ProTools)*. IEEE, 2020, pp. 19–26.

[8] S. R. Walli, "The posix family of standards," *StandardView*, vol. 3, no. 1, pp. 11–17, 1995.

[9] M. Factor, K. Meth, D. Naor, O. Rodeh, and J. Satran, "Object storage: The future building block for storage systems," in *2005 IEEE International Symposium on Mass Storage Systems and Technology*. IEEE, 2005, pp. 119–123.

[10] J. Wu, L. Ping, X. Ge, Y. Wang, and J. Fu, "Cloud storage as the infrastructure of cloud computing," in *2010 International Conference on Intelligent Computing and Cognitive Informatics*. IEEE, 2010, pp. 380–383.

[11] M. E. Pierce, M. A. Miller, E. H. Brookes, M. Wong, E. Afgan, Y. Liu, S. Gesing, M. Dahan, S. Marru, and T. Walker, "Towards a science gateway reference architecture," 2018.

[12] Transloadit. (2013, https://tus.io/protocols/resumable-upload.html) Tus protocol. [Online]. Available: https://tus.io/protocols/resumable-upload.html

[13] MINIO. (2021, https://min.io/) Object storage. [Online]. Available: https://min.io/

[14] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*, 2006, pp. 307–320.

[15] I. Foster, "Globus online: Accelerating and democratizing science through cloud-based services," *IEEE Internet Computing*, vol. 15, no. 3, pp. 70–73, 2011.

[16] B. Ross, E. Arslan, B. Zhang, and T. Kosar, "Managed file transfer as a cloud service," in *Cloud computing for data-intensive applications*. Springer, 2014, pp. 379–399.

[17] RClone. (2021, https://rclone.org/) Cloud sync. [Online]. Available: https://rclone.org/

[18] Alluxio. (2021, https://www.alluxio.io/) Data orchestration. [Online]. Available: https://www.alluxio.io/

[19] I. Ranawaka, S. Marru, J. Graham, A. Bisht, J. Basney, T. Fleury, J. Gaynor, D. Wannipurage, M. Christie, A. Mahmoud *et al.*, "Custos: Security middleware for science gateways," in *Practice and Experience in Advanced Research Computing*, 2020, pp. 278–284.