CHAPTER 8

MANAGING DISTRIBUTED SCIENTIFIC WORKFLOWS WITH GLOBUS

Kyle Chard
University of Chicago, Department of Computer Science, Chicago, IL, USA
chard@uchicago.edu
J. Gregory Pauloski
University of Chicago, Department of Computer Science, Chicago, IL, USA
jgpauloski@uchicago.edu
Ryan Chard
Argonne National Laboratory, Data Science and Learning, Lemont, IL, USA
rchard@anl.gov
Ian Foster
Argonne National Laboratory, Data Science and Learning, Lemont, IL, USA
foster@anl.gov

ABSTRACT: Scientific workflows increasingly span remote computing resources, from local desktops and scientific instruments to supercomputers, clouds, and AI accelerators. This distribution is driven by the nature of modern data-driven research and the availability of specialized computing hardware. Distribution creates new opportunities to improve performance and efficiency by exploiting resource heterogeneity and locality; however, it also creates new challenges related to portability and security. In this chapter, we describe Globus, a platform designed to tackle these challenges via a hybrid model in which cloud services securely manage the remote execution of arbitrary research activities. We describe how Globus Flows, a cloud-hosted workflow platform, combined with Globus Compute and Globus Transfer, enables researchers to define and execute workflows across diverse distributed computing resources. We present several example applications in real-time instrument analysis, simulation campaigns, and distributed model training that demonstrate how Globus addresses challenges in real-world scenarios.

8.1 Introduction

Modern science is inherently distributed, encompassing instruments, sensors, data repositories, specialized computing resources, archival storage, and publication repositories. The widespread adoption of distributed cyberinfrastructure is driven by several factors, such as the expanding scale of research, need to leverage specialized research instruments, growing data volumes, adoption of machine learning techniques, and global collaboration. As a result, many research processes must now be seamlessly deployed across diverse and distributed resources.

Workflows offer an effective way to represent and manage sophisticated research processes; however, adapting workflow methods designed for a single resource to a distributed environment introduces new challenges. For example, workflows must be able to orchestrate a variety of tasks – such as data movement and computation – across diverse computing resources, each with its own authentication and authorization policies, network and firewall configurations, data and computing interfaces, and computing environments. Research processes may also extend to other services, therefore requiring that workflows be able to interact with a diverse ecosystem of entities and services, such as collecting data from an instrument, calling web services to assign persistent identifiers, publishing data in domain-specific data repositories, and many others.

There are many additional benefits of adopting workflows to manage distributed applications, for example, enabling tasks within a workflow to be executed where they may be best suited, reducing latencies by executing tasks close to data, optimizing energy consumption by placing tasks on the most optimal resources, and enhancing resilience by failing over to other resources in situations of unavailability. These benefits can not only streamline research processes, but decrease costs, increase performance, and ultimately reduce the time to discovery. For example, timely access to resources can mean the difference between the success and failure of time-critical experiments [1], such as those used to steer a physical experiment, guide simulation campaigns, or even determine which experiment to perform.

In this chapter, we describe the Globus platform [2, 3] and outline how it enables secure, scalable, and performant distributed workflows. Globus offers a unique hybrid computing model in which a cloud-hosted platform securely and reliably orchestrates actions across an ecosystem of connected computing resources. We specifically describe how Globus Compute and Globus Transfer establish a global data and computing fabric, effectively reducing the barriers to using remote resources via standard interfaces, authentication methods, and approaches. We then describe Globus Flows, a hosted workflows platform, and outline how it allows researchers to define and deploy sophisticated workflows (called *flows*) that manage the execution of arbitrary resource processes on remote computing resources. We conclude by describing several real-world application examples – in materials science, real-time instrument science, and federated learning – and outlining how they benefit from Globus' workflow capabilities.

8.2 Globus

Globus was first released in 2011 to support remote data management and movement [4]. Over the past decade, the Globus platform has grown to deliver a broad range of critical research capabilities: Globus Transfer, for data transfer, synchronization, and sharing [2]; Globus Compute, for remote execution of computing functions [5]; Globus Flows, for managed cloud-hosted workflows [6]; Globus Search, for scalable, access-controlled indexing [7]; and Globus Auth, for standards-based authentication and authorization [8].

As shown in Figure 8.1, Globus adopts a unique hybrid cloud model in which the cloud-hosted platform, made up of various services, manages and orchestrates actions that are performed on remote computing resources. Remote resources are made accessible to Globus via lightweight software agents for data and computation management. These agents provide common interfaces to access local resources (e.g., a file system or batch scheduler) and implement a rich authentication and authorization model via which user actions are permitted

according to the authorization policies of the remote resource.

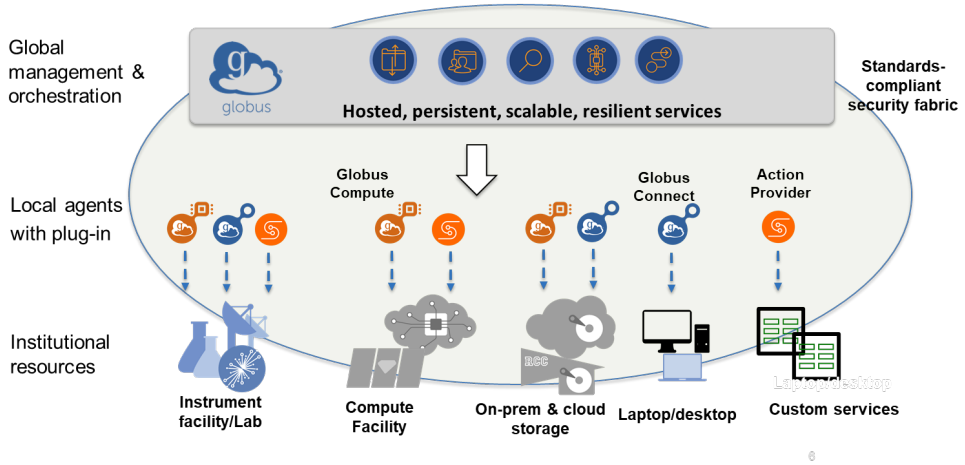

Figure 8.1: Globus hybrid architecture showing the Globus platform orchestrating actions on remote computing resources using software agents deployed on those resources or by using action providers that leverage other external services.

Globus services are implemented as REST services deployed on Amazon Web Services (AWS). Each with a similar JSON-based API. Most capabilities are available via Python and JavaScript Software Development Kits (SDKs), enabling developers to easily use capabilities within their Python programs, and a Command line interface (CLIs) that be installed and used to perform actions directly from within terminals. Most Globus services are accessible via a hosted Globus web application available at `https://globus.org`.

In the following sections, we describe how Globus supports distributed workflows; however, before discussing these capabilities, we must first describe Globus Auth [8] – the powerful authentication and authorization fabric on which all Globus capabilities depend. Globus Auth is, at its core, an identity and access management platform that has been designed to address the unique needs of scientific computing use cases. It acts as an identity broker to thousands of supported identity providers (IdPs) enabling users to authenticate using an external identity (e.g., institution, Google, ORCID). Given that researchers may have many identities, for use in many different scenarios, Globus Auth offers a unique identity federation model in which users can "link" many identities into a set. The set of identities can then be presented to a resource server, who can in turn make an authorization decision based on the identities in the set. Advanced high assurance (HA) policies can be employed by resource servers to require specific identities be used to access a resource, login within a particular session or time interval, and even characteristics of the authentication method (e.g., using two factor authentication).

Globus Auth implements the OAuth 2 and OpenID Connect specifications to enable standards-compliant use by external services and applications. Globus Auth is used by all Globus services; however, it can also easily be used by developers in their applications and services to outsource authentication and authorization functionality to Globus Auth. To do so, users must first register a Globus Auth resource server, defining a *scope* for that resource server. They can then configure their resource server with those credentials. Globus Auth supports

various OAuth workflows, enabling users to grant a third-party application limited access to a *resource* without sharing their login credentials directly. These workflows enable users to authenticate once, providing a unique "authorization code" to the resource server, and allowing the server to acquire access tokens that can subsequently used to access a resource on their behalf. Globus Auth extends OAuth with a novel dependent token model, via which a developer can state that their service is dependent on access to other services (i.e., their scopes). For example, a data management portal may depend on Globus Transfer to move data to/from the portal. After configuring a resource server with a dependent scope, authenticating users will be prompted to consent that the server can act as the user both within the scope of the resource server and for any dependent scopes of that resource server. This dependency model is a critical building block that enables creation of secure workflows that span resources in diverse administrative domains (see Section 8.5).

8.3 Globus Compute: A Global Compute Fabric

Distributed workflows require a common task representation and a mechanism to enable execution of those tasks on remote resources. Globus Compute provides these capabilities via a hybrid function-as-a-service (FaaS) model that establishes a reliable, secure, scalable, and high-performance function serving ecosystem on top of existing compute infrastructure [5, 9]. The hybrid architecture combines a cloud-hosted service for function management with a federated ecosystem of user-managed endpoints for function execution.

8.3.1 Globus Compute Model

In the FaaS model, users register programming functions and then invoke them without regard for the underlying infrastructure. Cloud-hosted FaaS platforms, like Amazon Lambda and Google Cloud Functions, provision infrastructure in their data centers using containers. Users therefore see only the execution of their function and have no knowledge of the physical or virtual resources on which that function executes. Globus Compute extends this paradigm to a federated model in which users can execute functions on their own, or other distributed, infrastructure; however, the execution of functions is still decoupled from infrastructure management.

Globus Compute supports the execution of Python *functions* on remote resources. *Any* Python function can be registered as a Globus Compute function. However, as is common in most FaaS platforms, all Python dependencies must be defined within the function body and the execution environment must have all required Python and system dependencies installed. While Globus Compute only supports Python functions, other software, scripts, or programs can be invoked as subprocesses within a Python function. Functions (their name and serialized body) must be registered with the Globus Compute service before use. In some cases, when using the Globus Compute client, function registration is performed transparently when a function is executed. Permissions – who may view or invoke the function – can also be specified during registration.

Compute resources are represented in Globus Compute as *endpoints*. Globus Compute supports two types of endpoints: single-user and multi-user endpoints. These endpoints may be deployed and managed by users or administrators of a system and can be configured for different resource types, such as laptops, clouds, or HPC clusters. When configuring an endpoint, users register it with the Globus Compute service, following a secure pairing

```python
from globus_compute_sdk import Executor

def double(x):
    return x * 2

with Executor(endpoint_id='9f6eb18f-f9a7-471b-a0c9-f12e1618adc4') as gce:
    future = gce.submit(double, 7)

    print(future.done())    # Prints False if not yet complete, True otherwise

    print(future.result())  # Blocks until the result is complete and prints 14
```

Listing 8.1: The Globus Compute `Executor` is an asynchronous, futures-based interface for executing remote functions.

process, after which functions can be submitted for execution on that endpoint by authorized users.

Globus Compute offers a "fire-and-forget" model in which a user requests that a function be executed and the cloud service manages the task execution and stores results for the user to retrieve asynchronously. The Globus Compute service handles the complexities of securely communicating function payloads to remote endpoints, monitoring execution and optionally retrying in case of failure, and storing eventual results in the cloud until retrieved by the user. The endpoint handles the provisioning of resources, function execution, and communication with the cloud service.

Globus Compute functionality is exposed via a REST API with various wrapper interfaces. For example, the Python Software Development Kit (SDK) provides a `Client` abstraction that can be used to authenticate with the service; register or remove functions, endpoints, and containers; execute functions; and retrieve function results. The SDK also provides an `Executor`, a subclass of Python's `concurrent.futures.Executor`, that provides a high-level interface for asynchronously executing functions. As demonstrated in Listing 8.1, a function and its arguments can be submitted to the executor which returns a `Future` representing the eventual result of the function. The future can be introspected to obtain the status of the task and can be called in a blocking mode to wait for the result.

8.3.2 Globus Compute Architecture

At the core of Globus Compute is a highly available, reliable, and secure cloud-hosted management service. The service is architected to horizontally scale and such that the actions of one user cannot interfere with another. The cloud service includes a function and endpoint registry, a REST API, and task/result queues. The registry maintains information on endpoints, functions, and users. Some information is also stored in a Globus Search index to provide discoverability of endpoints/functions. In addition to function/endpoint registration and management, the REST API is used to execute, monitor, and retrieve functions and their results. The API is secured using Globus Auth. When an endpoint registers with the service, two hosted RabbitMQ queues are created: for sending tasks and receiving results. The task queue is used to store submitted tasks which are retrieved by endpoints using a pull-based method. Results are sent from the endpoint to the endpoint's result queue. Tasks are queued at multiple levels, within the cloud service, at the endpoint, and on workers, to support reliable function execution, even on unreliable endpoints.

8.3.3 Globus Compute Endpoint

An endpoint is created by installing the Globus Compute agent software – a lightweight Python program. Globus Compute supports two types of endpoints: single-user endpoints (SEP) or multi-user endpoints (MEP) [10]. For SEPs, a user can install the endpoint themselves in user space. MEPs are designed to be installed by administrators as a privileged user for use by many users. MEPs essentially fork a new SEP for each user request. MEPs can be configured with various authorization policies, e.g., regarding suitable user identities and session life times. Importantly, they can also be configured with an extensible identity mapping approach that maps the Globus Auth-provided identity used for authentication with the local user account in which the MEP forks the SEP process. The same identity mapping approach is also used by Globus Transfer.

Globus Compute endpoints can be configured to use a diverse range of compute resources. This flexibility is achieved through the use of Parsl [11], a parallel programming library for Python. Globus Compute uses Parsl's pilot job model and its execution provider abstraction. Parsl implements a common interface for provisioning resources from various types of systems. It includes providers for local execution, batch scheduler systems (e.g., Cobalt, Slurm, PBS), and cloud platforms (e.g., Kubernetes). Each provider exposes extensive configuration options for resource-specific requirements such as allocations, resource types, queues, and limits. Globus Compute extends Parsl's High-Throughput Executor (called the GlobusComputeEngine) as a general-purpose pilot job model for managing a pool of *workers* deployed on provisioned resources. Briefly, for each provisioned node, the GlobusComputeEngine will deploy a *manager* process and a set of *worker* processes (the number of workers is defined in the endpoint configuration). The *workers* are independent processes capable of executing a Globus Compute task. Each *worker* communicates with the *manager* to receive tasks and return results. The *manager* in turn communicates with an *interchange* process deployed on the endpoint to pass tasks/results between the endpoint and the *workers*. The *manager* effectively reduces the number of network connections between *workers* and the endpoint.

Endpoints are responsible for managing the entire task execution process. For example they can deploy and manage compute environments (containers or virtual environments), retry tasks on failure, and dynamically scale resources in response to workload levels. The endpoint model cleanly separates compute resource configuration and management from application design – functions can be trivially executed on different resources by simply changing the endpoint ID to which tasks are submitted.

8.3.4 Container Management

One of the most significant challenges with the FaaS model is ensuring that the execution environment is suitable for executing a function. In Globus Compute, functions, by default, are executed within the worker's Python environment. The endpoint allows the worker environment to be configured in many ways. For example, workers can be configured to load a system module, activate Python environments on initialization, or even install dependencies before starting. While such methods are commonly used on HPC batch systems, they do not enable portability between systems in most cases.

Globus Compute also supports the use of containers by allowing the specification of containers in the endpoint configuration. Containers ensure that the dependencies of Globus Compute and registered functions are available regardless of the default environment of

the endpoint. In HPC environments, the endpoint deploys workers in containers. Globus Compute supports various container technologies (e.g., docker and apptainer) and different routing mechanisms to send tasks to specific containers. Globus Compute also supports the use of containers in Kubernetes. Here both the manager and the worker are deployed within a pod and thus the manager cannot change the type of worker container. In this case, a set of managers are deployed with specific container images and the Globus Compute endpoint simply routes tasks to corresponding managers (matching their types).

8.3.5 Globus Compute for Workflows

Globus Compute implements a global fabric for executing tasks on arbitrary computing resources. It's simple REST API allows for workflow systems to easily use it to execute components of a workflow. We describe below how Globus Flows leverages Globus Compute to execute tasks in distributed workflows. Parsl, a more traditional single-site scientific workflow management system, also provides support for using Globus Compute to execute tasks. In this case, a Parsl program, written in Python, can configure a GlobusComputeExecutor with a specific endpoint ID. After authentication, tasks in Parsl's DAG are dispatched for execution via a call to Globus Compute. The advantage of this approach is that the Parsl workflow can run on one machine (e.g., a laptop), and the tasks of the workflow can run on another (e.g., a cloud or cluster). While Globus Compute currently requires that users specify a specific endpoint ID on which to execute a task, we have developed methods to automatically send tasks for execution on the best of a set of possible endpoints [12, 13].

8.4 Globus Transfer: A Global Data Fabric

Distributed workflows require a common way to access and manage distributed data irrespective of the storage system, interface, or protocol used. Globus Transfer is a cloud-based platform that supports remote data access, management, transfer, and sharing across an ecosystem of more than 60,000 active Globus endpoints. Following the hybrid deployment model, administrators or users first deploy Globus Connect software on a resource to make it accessible to the Globus data ecosystem. Authorized users can then access and manage data on remote endpoints via the cloud-hosted service. If enabled, users can upload/download data directly and securely using HTTPS, or they can transfer data directly between two different endpoints using the GridFTP protocol [14].

8.4.1 Globus Connect

Globus Connect is distributed in two versions: Globus Connect Personal is designed to be installed for single users on Mac, Windows, and Linux computers; Globus Connect Server is a multi-user package that can be installed on Linux servers and offers capabilities to manage users, map identities to local (or remote) accounts, serve data with HTTPS, distribute deployment across several nodes, and implement high assurance policies.

When a user or administrator installs Globus Connect they create a Globus Transfer *endpoint*. In some scenarios, for example on HPC systems with many Data Transfer Nodes (DTNs), Globus Connect Server may be deployed on several nodes. These nodes are grouped as part of a single endpoint and workload is distributed among the nodes. Globus Connect Server endpoints support management capabilities for mapping Globus Identities to local

users or credentials for external storage providers (e.g., Google Drive) and for configuring and managing *storage gateways* for connecting to different storage systems (see Section 8.4.2).

Users interact with endpoints via *collections*, either a *mapped collection* or a *guest collection*. A mapped collection provides access to a storage system using the credentials of the logged-in user. That is, the user is "mapped" to an underlying credential or access mechanism. For example, the user "bob@globus.org" may be mapped to a local user "bob" on a POSIX storage system or it may be mapped to an API key for accessing Google Drive. Globus Connect Personal is preconfigured to use a mapped collection to the local file system as the owner of the endpoint.

A guest collection may be created by a user on a mapped collection. The guest collection subsequently provides access using the credentials of the user who created it (not the user who is accessing it). The owner can set access control permissions that are used to determine what other users are permitted to do (read or write specific paths) and manage (add other permissions) that collection. Thus, guest collections allow users to directly share access to data without needing to move those data to a cloud-hosted location.

8.4.2 Storage Gateways

The storage landscape is increasingly diverse, combining traditional POSIX-based file systems with various other types of storage, such as archival tape storage and cloud object storage. Globus Connect is designed to support these different storage systems via an extensible *connector* model. Globus Connect supports thirteen non-POSIX connectors, including for cloud storage (e.g., Box, Google Drive), object storage (e.g., Ceph), archival storage (e.g., HPSS, Spectra Logic Black Pearl), and others (e.g., iRODS). When configuring Globus Connect Server, administrators must first define a storage gateway. This gateway defines the connector to be used and the access policies for that storage system. For example, the gateway can define authentication requirements and a set of paths to which it provides access.

8.4.3 Globus Transfer Service

Globus Transfer is operated as a cloud-hosted service deployed on AWS. The service essentially acts as a data access and transfer orchestrator, managing data access and movement between endpoints. Users interact with the service through various interfaces, including the Globus web application, command-line interface (CLI), REST APIs, and Python SDK. These interfaces communicate with the cloud service to authenticate users, perform transfers, and provide real-time status updates.

The Globus Transfer service is responsible for managing transfer requests, monitoring transfer progress, handling retries, and ensuring data integrity. The Globus Transfer architecture separates the control and data planes to enhance security and performance while providing users with a seamless and user-friendly experience. That is, the service uses a control channel to communicate with endpoints; however, all data moves directly between endpoints rather than via the cloud service (often referred to as "third-party" data transfer). Globus Transfer performs some operations over the control channel, for example, directory listing, as this information is relied upon only by the client.

The Globus Transfer service treats transfer requests as "jobs" that it manages. When a user submits a transfer request, they select the two collections (source and destination) and

paths (files or directories) on each collection. The transfer service communicates with each endpoint, validating that the user is authorized to access the collection. This authorization process combines cloud-hosted policies with final authorization decision points exposed by the endpoint. The transfer service instructs the two endpoints to establish a connection between each using the GridFTP protocol. Various heuristics are used to define the connection, number of concurrent streams, parallelism, pipelining, etc. These parameters are tuned to obtain high performance transfers between the endpoints, based on their capabilities. Globus Transfer provides a fire-and-forget transfer model. A user can start a transfer and leave the transfer service to deal with the complexities of the transfer, for example by checking the integrity of transferred files by comparing checksums and re-transferring data when needed, waiting for endpoints to recover from failures, and managing authentication renewal according to local policies.

8.4.4 Globus Transfer for Workflows

Workflows that seek to orchestrate task execution on one or more remote systems, typically need to manage data movement. Data to be moved may include input or output files, intermediary data shared between tasks, or even task definitions, computing environments, and containers. Workflow systems have long supported different data staging mechanisms, often using SCP, rsync, or HTTP. While straightforward, one challenge with many existing implementations is that they support only direct data upload/download and thus require that sender/receivers can directly access one another. In practice, such connectivity is rare, as for example, inbound connections to personal computers are blocked by firewalls and are complicated by NAT, while direct HTTP connections between HPC systems are often not permitted. For example, many systems will block inbound HTTP connections, while SSH connections typically require two factor authentication and systems may restrict agent forwarding.

Globus Transfer provides an ideal solution as it can coordinate third-party transfers directly between systems and is able to overcome the various authorization challenges. For example, it uses TCP hole-punching and a cloud-hosted relay server to establish connections between endpoints behind NAT or firewalls. Globus Transfer plays a critical role in the cloud-hosted workflows described below as it allows data to be seamlessly moved between different locations. Globus Transfer is also used by many other workflow systems such as Parsl and Pegasus [15]. For example, Parsl allows users to configure a Globus Transfer data stager that will move input, output, and intermediary data between the Parsl client and workers.

8.5 Globus Flows

Globus Flows [6, 7] is a cloud-hosted workflows platform that automates execution of complex sequences of tasks, known as *flows*, across distributed environments. These flows can involve data transfers, computations, and other operations that need to be executed in a coordinated manner. As a cloud-hosted service, Globus Flows allows users to compose and then outsource execution of their flows without needing to maintain persistent services, detect and recover from failures, or retrieve results synchronously even when flows span geographic locations and administrative domains. The service is designed to handle both the spatial and

temporal complexities of research workflows, making it a versatile workflow platform for a wide range of scientific applications.

At the core of Globus Flows is its ability to define flows using a declarative, JSON-based representation. Users specify the series of *actions* that make up a flow, along with the conditions under which these actions should be executed. Flows may include different control sequences, such as loops and conditions, enabling representation of different workflow patterns, such as map-reduce. Actions are implemented by *action providers*, which are modular services that perform a specific task such as data transfer, computation, or metadata management. The flexible nature of this architecture allows flows to integrate with a wide array of external services and tools, making it possible to define flows that involve a diverse range of tasks. Users can build new action providers or integrate existing services as an action provider by implementing the action provider API. This modularity and extensibility ensures that workflows can be easily adapted or extended to accommodate new requirements as research projects evolve.

8.5.1 Action Providers

Action Providers serve as a standardized interface to integrate custom tasks – represented by web services – into the Globus ecosystem. The action provider interface assumes an asynchronous action and requires functionality to start an action, retrieve the status or result of that action, cancel an action, and optionally release any state associated with that action. Synchronous actions can be easily integrated with this asynchronous API. The API also requires an interface to introspect the action and discover information about the action and required input/output schema. Actions may consume and return JSON formatted data.

Action Providers are implemented as a Globus Auth Resource Server and have their own associated scope. This approach, building on the OAuth 2 protocol, allows a flow to be defined with a set of required action scopes. It therefore allows the flows service to negotiate consent with a user to enable the flow to perform all actions on a user's behalf. Flows may execute actions either as themselves (i.e., as the flow) or as the user who started the flow. This rich and flexible authorization model allows flows to be used in a diverse range of use cases and is particularly focused on supporting the needs of flows that span geographic and administrative boundaries. For example, it allows a flow to transfer data between two endpoints for which only a specific user is permitted to access.

Globus hosts a set of Action Providers that enable data management with Globus Transfer, execution of compute tasks using Globus Compute, indexing and discovery of data in Globus Search, association of persistent identifiers using DataCite, and others. To simplify the creation of Action Providers, we have created a FastAPI-based toolkit that streamlines the development process. This toolkit allows developers to quickly build and deploy action providers using a framework that can be used to create the required interface, integrate with Globus Auth, and simplify many of the underlying complexities of creating actions.

8.5.2 Flows Service

The Globus Flows service is implemented as a REST service deployed in Docker containers on AWS Elastic Container Service (ECS). Rather than develop a new workflow system, the Flows service builds on Amazon Step Functions (ASF) – a cloud-based workflow system designed and operated by AWS. Globus Flows extends ASF with implementation of the Flows

authentication and authorization model, asynchronous Action Providers, and support for science use cases. Briefly, the Flows service takes a user-provided Flow definition, registers it with Globus Auth, and deploys a state machine with ASF. The deployed state machine is a translated representation of the Flows definition and includes ASF "Task" states for each action to be called. Each invocation of an action is sent to a cloud-hosted queue which in turn uses an AWS Lambda function to invoke the remote action and also to poll the action for completion. The Flows service manages the execution of the workflow, catching transitions from ASF and implementing error detection and retry logic. Results are stored in a cloud-based database, enabling users to review flow progress, inspect results, and rerun flows.

8.5.3 Defining and Running Flows

Listing 8.2 shows an example definition for a flow that will tar a dataset on one endpoint and then transfer the tar file to another endpoint. The definition highlights the two states in the flow, in this case two actions (`TarFiles` and `TransferTar`), and the transitions between those states (noted with the "Next" attribute). Each state is passed a set of parameters, these can be static or dynamic and may be passed in by the user as part of the initial *context* or may be added to the context by prior steps of a flow. For example, in this flow users must supply the Globus Compute endpoint ID and the ID of the registered tar function, the source path and tar file name, and the destination endpoint and path. Note that the source path for the transfer is computed from the result of the tar compute function (Line 28).

Many flows will require custom input data when started. To simplify use by other users, flow authors can define an input schema for the flow. The input schema is registered with the Flows service and is used to validate the input document supplied by a user when starting a flow. It is also used by the Globus web application to render a graphical user interface to guide users when running the flow.

After defining the flow and registering it with Globus Flows, it is then made visible to a set of authorized users and runable by another set of authorized users. By default, visibility and runability is restricted to the user who defined the flow. To run the flow, the user may supply a JSON document as input as required by the flow. If an input schema is associated with the flow, the input document provided by the running user will be validated by the Flows service. Listing 8.3 shows an example input document that can be used to run the flow defined above.

After running the flow, the user who ran the flow (or other user who are authorized to view the status of the flow) can monitor the status of the flow and each action. Figure 8.2 shows the flow status for a completed flow in the Globus web-based interface. The interface shows when each action starts and completes as well as the time taken for each task. After successful or failed completion of the flow the user can review logs from the flow, state passed between steps, and in the case of errors, reports from each of the steps.

8.6 Application Examples

We describe three types of distributed workflows that use Globus. These examples are real-time analysis of data from scientific instruments, training machine learning models using federated learning, and managing simulation campaigns.

```
flow_definition = {
  "StartAt": "TarFiles",
  "States": {
    "TarFiles": {
      "Next": "TransferTar",
      "Type": "Action",
      "ActionUrl": "https://compute.actions.globus.org/v3",
      "Parameters": {
        "tasks": [
          {
            "kwargs": {
              "src_path.$": "$.src.path",
              "dest_path.$": "$.tar_name"
            },
            "function_id.$": "$.compute.function_id"
          }
        ],
        "endpoint_id.$": "$.compute.endpoint_id"
      },
      "ResultPath": "$.compute_result"
    },
    "TransferTar": {
      "Type": "Action",
      "ActionUrl": "https://transfer.actions.globus.org/transfer",
      "Parameters": {
        "DATA": [
          {
            "source_path.=": "compute_result.details.result[0]",
            "destination_path.$": "$.dest.path"
          }
        ],
        "source_endpoint.$": "$.src.endpoint_id",
        "destination_endpoint.$": "$.dest.endpoint_id"
      },
      "ResultPath": "$.transfer_to_dest_result",
      "End": True
    }
  }
}
```

Listing 8.2: Tar and transfer flow that shows a compute function executing the tar task on a remote resource before transferring the tar data to another endpoint.

```
{
    'compute': {
        'endpoint_id': '4b116d3c-1703-4f8f-9f6f-39921e5864df',
        'function_id': '93fa16e3-0c93-4c43-a8d2-fabd1a1a74d5'
    },
    'src': {
        'endpoint_id': '6c54cade-bde5-45c1-bdea-f4bd71dba2cc',
        'path': '/path/to/tar/'
    },
    'tar_name': 'output.tar'
    'dest': {
         'endpoint_id': '31ce9ba0-176d-45a5-add3-f37d233ba47d',
         'path': '/~/output.tar'
    }
}
```

Listing 8.3: Input for the Tar and Transfer flow, specifying the endpoints to use, function identifier, and path to act on.

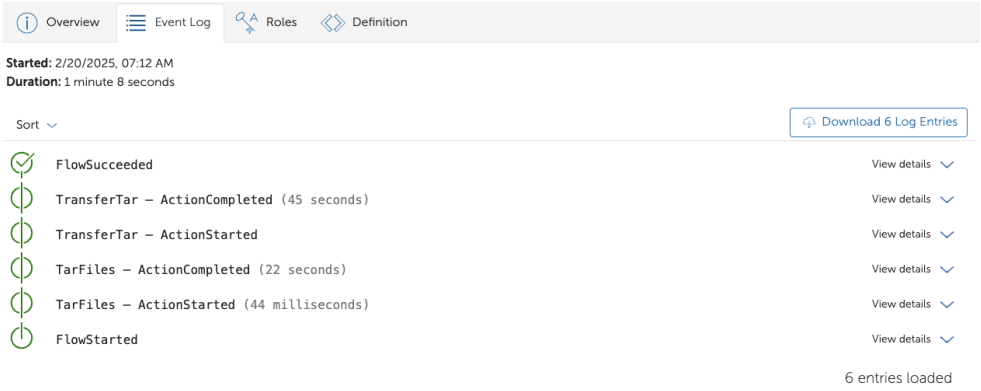

Figure 8.2: The Globus Flows UI showing the event log of the Tar and Transfer flow. Each task can be inspected to review outputs or debug failures. The flow access control roles and definition are available in other tabs.

8.6.1 Linking Computing and Scientific Instruments

Linking scientific instruments with HPC resources enables researchers to efficiently manage and interpret data, enabling real-time analysis and accelerating the pace of discovery [16]. Argonne National Laboratory's Advanced Photon Source (APS) recently underwent an $815 million upgrade to enhance the electron storage ring and X-ray beamlines. This upgrade will significantly increase the volume of data generated, with estimates that data volumes will grow by at least two orders of magnitude over the next decade, producing hundreds of petabytes of raw data each year. To deal with this data explosion, tens of petaflops of both on-demand computing resources and traditional batch processing capabilities will be required, necessitating advanced workflows to automatically filter and process data as they are acquired.

Many APS beamlines now use Globus Flows to connect with the on-demand computing resources of Argonne's Polaris supercomputer at the Argonne Leadership Computing Facility (ALCF). Over 30 workflows, spanning nine scientific applications, have been developed and deployed for operational use. APS beamlines use these workflows during real-time experiments to produce new scientific results [17]. These workflows combine Globus Flows, Globus Compute, and the APS Data Management System [18]. The APS Data Management System integrates with beamline instruments to oversee data acquisition, manage raw data, and orchestrate analysis. The system uses Globus to automatically perform analysis and publication flows, transfer data between APS and ALCF, and execute computational workflows on remote supercomputers. This integration allows instruments to automatically perform complex workflows, transferring data collected at beamlines to remote computing resources for rapid analysis, with results promptly returned to the experimentalist. A Globus Group, including the identities of users involved in a particular experiment, is used to control authorization and enables data to be automatically published to a data portal and securely shared with experiment members.

These workflows have highlighted the need for infrastructure that supports real-time distributed workflows. The ALCF supercomputers have deployed Globus Connect and Globus Compute agents to make data and compute accessible to the Globus platform. Working with

ALCF, they have implemented on-demand queues for immediate execution of jobs, service accounts dedicated to APS instruments to streamline workflow deployment and operation, and resilient endpoint nodes for workflow orchestration and execution. Additionally, data transfer between the APS and ALCF is facilitated by Globus Transfer and a 1.6 Tbps network, designed following the ScienceDMZ pattern [19].

Globus is also used at the Advanced Light Source (ALS) for automated analysis workflows, enhancing the efficiency and effectiveness of its operations. An ALS tomography beamline uses a Prefect workflow to process every dataset that is acquired. This workflow uses Globus Transfer to move data to Argonne, where Globus Compute is used to perform analysis on the Polaris on-demand queue. This integration allows for the rapid and secure processing of data, ensuring that the results are promptly returned to the experimentalist at the beamline. The Prefect workflow is deployed in production to act on every dataset captured at the beamline. This model showcases how Globus can be easily incorporated into existing workflows to securely and reliably outsource data management tasks across distributed resources.

8.6.2 Distributed and Decentralized Learning

Traditionally, training a machine learning (ML) model on distributed data required first transferring all data to a centralized resource (e.g., HPC cluster). *Federated learning* (FL) [20, 21] is a new distributed ML training paradigm that overcomes the need to centralize data. In FL, separate models are trained directly where data reside (e.g., on instruments, edge devices, sensors). A FL workflow collects these distributed models and aggregates them, typically using a central aggregator, to create a global model that is shared with participating devices and used for the next round of training. This approach enables devices to benefit from the data of other devices, but, most importantly, without ever revealing or moving raw data. Figure 8.3 illustrates the FL training process.

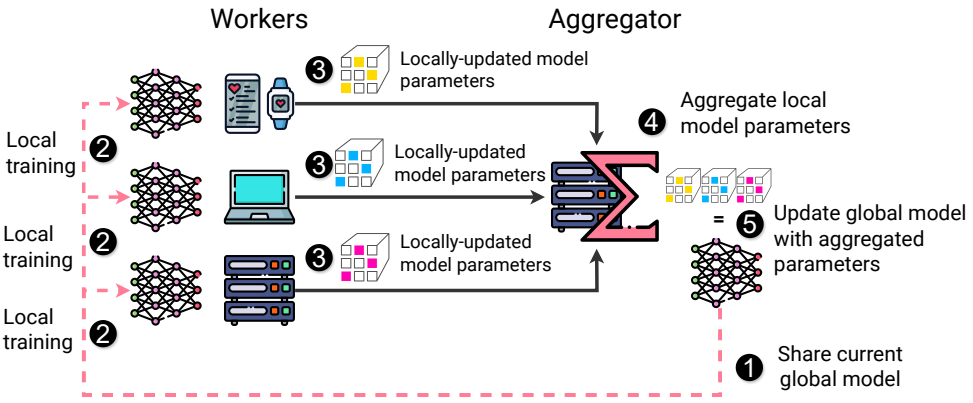

Figure 8.3: High-level view of a FL workflow. FL workers, deployed on distributed devices, collect data and train local models. The Aggregator combines these local models and distributes a global model to edge devices for use or further training. Figure reproduced from [22].

Despite the obvious benefits of FL in science, there are significant barriers that prohibit the practical deployment of FL. Existing FL frameworks [23, 24] can be difficult to deploy, configure, and use; do not support the diverse and heterogeneous cyberinfrastructure and topologies found in science; and can be inefficient/insecure as they require data and models be moved through a central entity. Furthermore, much FL literature (e.g., new aggregation algorithms, FL frameworks, and use cases) conducts experiments using a single computer which fails to capture the complexities of real distributed scientific deployments.

Three Python-based frameworks, APPFL (Advanced Privacy Preserving Federated Learning) [25], Flight (Federated Learning in General Hierarchical Topologies) [22] (previously called FLoX [26]), and Academy [27] use the Globus platform for deployment of FL in real-world environments. These frameworks have been broadly applied in science applications, including biomedicine [28], precision agriculture [29–31], astrophysics, and materials science.

APPFL is an open-source Python library for deploying FL workflows with advanced privacy-preserving techniques. APPFL supports FL across institutions ("cross-silo") with the ability to scale on distributed, heterogeneous computing resources to create robust, trustworthy AI models. APPFL's modular architecture with six user-customizable components – aggregator, scheduler, trainer, privacy, communicator, and compressor – enables different FL strategies, workflows, and communication protocols to be used. The *aggregator* supports popular algorithms to aggregate local models; the *scheduler* uses various server-side scheduling algorithms to handle different arrival times of local models; the *trainer* supports multiple local trainers; the *privacy* component supports global and local differential privacy schemes; the *communicator* supports Globus Compute for secure distributed training across deployed Globus endpoints; and the *compressor* supports several lossy compressors.

Flight is an open-source Python library for implementing FL workflows across heterogeneous and distributed devices ("cross-device"). Flight supports FL workflows on local resources (using Parsl) and over wide area networks (using Globus Compute and ProxyStore [32, 33], which in turn can use Globus Transfer). Users define a topology of trainers and aggregators. For each device, the topology specifies the Globus endpoints deployed on that device. Flight then manages the training workflow across the topology by dispatching training and aggregation tasks to each device, moving trained and aggregated models between devices, and distributing the global model for use and further training. Experiments have shown that Flight can scale multi-node performance beyond state-of-the-art frameworks like Flower and its hierarchal topologies can reduce data communication (by up to 60%) compared to centralized examples.

Academy is a middleware that supports the creation and deployment of autonomous agents across federated research infrastructure, including HPC systems, experimental facilities, and data repositories. By enabling large-scale, asynchronous collaboration among agents, Academy forms a robust foundation for decentralized learning. In this context, each agent operates independently – training a local model, exchanging model updates with neighbors, and performing aggregation – without requiring centralized coordination, and agents and their communication links form a graph structure, reflecting real-world network topologies. Experiments demonstrate scaling decentralized learning workloads to 1536 GPUs on the Aurora supercomputer, flexible deployment of agents with Globus Compute, and efficient data transfer via ProxyStore. These results highlight Academy's suitability for decentralized learning scenarios, where data locality, privacy, and communication overhead are key considerations, and centralized orchestration is impractical or undesirable. Similar workflows

can also be realized for other agentic discovery scenarios [34].

8.6.3 Simulation Campaigns

Large-scale computational science campaigns involve the execution of an ensemble of individual simulations on HPC resources [35, 36]. Individual simulations are selected from a search space of possible parameters and are executed in parallel. Ensemble simulations are steered by analyzing the results of prior simulations to determine which simulations to perform next. The simulation selection policies vary in complexity but increasingly rely on ML models that are re-trained throughout the campaign to optimize the simulation selection, a pattern referred to as *active learning*.

Active learning has proven an effective method for increasing scientific outcomes and optimizing resource efficiency, but these applications increasingly require heterogeneous resources. For example, simulation codes may be optimized for parallel execution on the many CPU cores of a cluster while inference and training of ML models is best suited for specialized AI accelerators. Colmena is a general purpose library for expressing the steering of ensemble simulations and executing workflows across heterogeneous compute resources [37].

In Colmena, a *thinker* controls what tasks (e.g., simulations, model retraining) to perform, what resources to execute tasks on, and how to allocate available resources between task types. The thinker is composed of agents which interact with each other to process task results, respond to events, re-allocate resources, and submit new tasks. Colmena effectively describes a workflow that maps task types to specific Globus Compute endpoints, enabling access to remote, heterogeneous resources that are optimized for specific task execution.

Colmena, Globus Compute, and Globus Transfer are used together to deploy multi-site applications for *molecular design* and *surrogate fine-tuning* [38]. The objective of the *molecular design* application is to find molecules with desirable properties from candidate dataset. In this case, the desirable property is high ionization potentials (IP) which are necessary for designing organic electrolytes. The active learning workflow selects an initial set of molecules to simulate on a CPU to determine the IP. A surrogate ML model is then trained on a GPU to predict the IP of a molecule. Surrogate models are many orders of magnitude cheaper than performing simulation, so the surrogate model is used to predict IPs for the entire candidate set. Molecules with the highest predicted IPs are then selected for simulation. The surrogate model is periodically retrained with the results of new simulations to improve the prediction accuracy. The *surrogate fine-tuning* application follows a similar active learning approach with the goal of producing a surrogate model for expensive quantum mechanics simulations that can predict the energies and forces on clusters of water surrounding a methane solute.

These applications highlight key challenges in deploying scientific workflows on heterogeneous hardware. Compute clusters are typically composed of homogeneous nodes, necessitating access to resources at multiple sites; authentication and resource provisioning mechanisms differ between sites; and execution across many sites increases the need for a reliable and robust management system. Globus Compute and Globus Transfer provide a global fabric that addresses these challenges and enables the execution of these two applications across multiple sites.

8.7 Related Work

The Globus platform adopts the cloud model that has been widely adopted in industry. Cloud providers like Google, Amazon, and Microsoft, provide a range of platform capabilities that target industry use cases. Globus builds upon these platform services to deliver capabilities designed specifically for scientific workflows. In the scientific domain, there are some related efforts such as Tapis [39] and the NERSC Superfacility API [40]. Unlike Globus, these services are designed to be deployed at a specific site and thus provide service capabilities oriented around that site. Both Tapis and Superfacility interoperate with Globus for data management.

The proliferation of workflow systems has been well-noted, with hundreds of seemingly comparable workflow systems available for use [41]. These workflow engines, many of which have been used in science [11, 15, 42–45] enable representation of complex applications and efficient execution on diverse cyberinfrastructure. The specific nature of the workflow systems differ, but most are designed to coordinate execution of programs, batch jobs, or web service calls. Task-based systems, such as Parsl [11], Pegasus [15], and Swift [46] manage the execution of a series of computational tasks, for example, by executing programming functions or by making calls to locally executable programs and scripts. Service-based systems, such as Globus Flows and Taverna [47] invoke actions via web service APIs.

Most workflow systems are designed to be deployed by a single user on a single system. Some, for example Parsl, are able to manage execution across various computing resources. However, Parsl does so by relying on SSH connections from a client machine to the execution environment and requires inbound network connectivity from the execution machine to the client. Workflow systems like Galaxy [43] are designed for multi-tenant deployments via which many users can deploy and manage workflows. Well-known instances of workflow systems, such as Galaxy, are used by large communities of users. However, in practice they primarily leverage co-located cloud resources that are associated with the deployment. That is, they do not readily manage execution on disparate computing resources.

Cloud computing providers, among their suite of provided services, are increasingly providing workflow-like capabilities for automation, development operations, and data wrangling. For example, AWS's Simple Workflow Service (SWF) [48] and and Step Functions (SFN) service [49] enable definition and execution of workflows on the cloud. GitHub Actions [50] and AWS CodePipeline [51] offer automation tools that directly support continuous integration and continuous deployment pipelines.

8.8 Summary

The increasingly distributed scientific research process requires new workflow systems that can manage diverse actions across disparate infrastructure securely, efficiently, and scalably. We have described how the Globus platform can satisfy the needs of modern distributed science using a hybrid model model that securely coordinates scientific workflows across remote endpoints. Globus Flows combines the convenience, availability, and usability of a cloud-hosted service to develop and deploy workflows that themselves leverage Globus services for managing data and computation across the computing continuum. Globus Transfer and Globus Compute effectively create a global data and compute fabric, respectively, that allows Globus Flows to dispatch tasks *anywhere* while maintaining a robust global authenti-

cation and authorization model and also enforcing local authorization policies. Future work in this area will build upon this general platform to further improve the usability and performance of the workflows ecosystem. For example, in ongoing work, researchers are using the Globus platform to dynamically select where to execute tasks based on various metrics including energy [12] and performance [13]. Such approaches can consider many different costs, such as HPC service units, monetary costs, energy use, and carbon, while also considering other measures of performance and reliability.

Acknowledgment

This work was supported by Argonne National Laboratory under Contract No. DE-AC02-06CH11357 and by NSF awards 2004894, 2209919, and 1835890.

Bibliography

[1] A. V. Babu, T. Zhou, S. Kandel, T. Bicer, Z. Liu, W. Judge, D. J. Ching, Y. Jiang, S. Veseli, S. Henke, *et al.*, "Deep learning at the edge enables real-time streaming ptychographic imaging," *Nature Communications*, vol. 14, no. 1, p. 7059, 2023.

[2] K. Chard, S. Tuecke, and I. Foster, "Efficient and secure transfer, synchronization, and sharing of big data," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 46–55, 2014.

[3] K. Chard, I. Foster, and S. Tuecke, "Globus: Research data management as service and platform," in *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, PEARC '17, (New York, NY, USA), Association for Computing Machinery, 2017.

[4] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, M. Link, S. Martin, K. Pickett, and S. Tuecke, "Software as a service for data scientists," *Communications of the ACM*, vol. 55, no. 2, p. 81–88, 2012.

[5] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, and K. Chard, "funcX: A federated function serving fabric for science," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, ACM, 2020.

[6] R. Chard, J. Pruyne, K. McKee, J. Bryan, B. Raumann, R. Ananthakrishnan, K. Chard, and I. T. Foster, "Globus automation services: Research process automation across the space–time continuum," *Future Generation Computer Systems*, vol. 142, pp. 393–409, 2023.

[7] R. Ananthakrishnan, B. Blaiszik, K. Chard, R. Chard, B. McCollam, J. Pruyne, S. Rosen, S. Tuecke, and I. Foster, "Globus platform services for data publication," in *Practice and Experience on Advanced Research Computing*, pp. 14:1–14:7, ACM, 2018.

[8] S. Tuecke, R. Ananthakrishnan, K. Chard, M. Lidman, B. McCollam, S. Rosen, and I. Foster, "Globus auth: A research identity and access management platform," in *Int. Conf. on e-Science*, pp. 203–212, 2016.

[9] Z. Li, R. Chard, Y. Babuji, B. Galewsky, T. J. Skluzacek, K. Nagaitsev, A. Woodard, B. Blaiszik, J. Bryan, D. S. Katz, *et al.*, "FuncX: Federated function as a service for science," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4948–4963, 2022.

[10] R. Ananthakrishnan, Y. Babuji, M. Baughman, J. Bryan, K. Chard, R. Chard, B. Clifford, I. Foster, D. S. Katz, K. Hunter Kesling, C. Janidlo, R. Mello, and L. Wang, "Enabling remote management of faas endpoints with globus compute multi-user endpoints," in *Practice and Experience in Advanced Research Computing 2024: Human Powered Computing*, PEARC '24, (New York, NY, USA), Association for Computing Machinery, 2024.

[11] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster, M. Wilde, and K. Chard, "Parsl: Pervasive parallel programming in Python," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '19, (New York, NY, USA), p. 25–36, Association for Computing Machinery, 2019.

[12] A. Kamatar, V. Hayot-Sasson, Y. Babuji, A. Bauer, G. Rattihalli, N. Hogade, D. Milojicic, K. Chard, and I. Foster, "GreenFaaS: Maximizing energy efficiency of HPC workloads with FaaS," 2024.

[13] R. Kumar, M. Baughman, R. Chard, Z. Li, Y. Babuji, I. Foster, and K. Chard, "Coding the computing continuum: Fluid function execution in heterogeneous computing environments," in *Int. Parallel and Distributed Processing Symp. Workshops*, pp. 66–75, 2021.

[14] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus striped GridFTP framework and server," in *ACM/IEEE conference on Supercomputing*, p. 54, IEEE Computer Society, 2005.

[15] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[16] R. Vescovi, R. Chard, N. D. Saint, B. Blaiszik, J. Pruyne, T. Bicer, A. Lavens, Z. Liu, M. E. Papka, S. Narayanan, N. Schwarz, K. Chard, and I. T. Foster, "Linking scientific instruments and computation: Patterns, technologies, and experiences," *Patterns*, vol. 3, no. 10, p. 100606, 2022.

[17] M. Prince, D. Gürsoy, D. Sheyfer, R. Chard, B. Côté, H. Parraga, B. Frosik, J. Tischler, and N. Schwarz, "Demonstrating cross-facility data processing at scale with laue microdiffraction," in *Workshops of the Int. Conf. on High Performance Computing, Network, Storage, and Analysis*, pp. 2133–2139, 2023.

[18] S. Veseli, N. Schwarz, and C. Schmitz, "Aps data management system," *Journal of Synchrotron Radiation*, vol. 25, no. 5, pp. 1574–1580, 2018.

[19] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The Science DMZ: A network design pattern for data-intensive science," *Scientific Programming*, vol. 22, no. 2, pp. 173–185, 2014.

[20] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, pp. 1273–1282, PMLR, 2017.

[21] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[22] N. Hudson, V. Hayot-Sasson, Y. Babuji, M. Baughman, J. G. Pauloski, R. Chard, I. Foster, and K. Chard, "Flight: A FaaS-based framework for complex and hierarchical federated learning," 2024.

[23] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and NicholasD.Lane, "Flower: A friendly federated learning framework," *arXiv preprint arXiv:2007.14390*, 2022.

[24] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 374–388, 2019.

[25] M. Ryu, Y. Kim, K. Kim, and R. K. Madduri, " APPFL: Open-Source Software Framework for Privacy-Preserving Federated Learning ," in *Int. Parallel and Distributed Processing Symp. Workshops*, pp. 1074–1083, 2022.

[26] N. Kotsehub, M. Baughman, R. Chard, N. Hudson, P. Patros, O. Rana, I. Foster, and K. Chard, "FLoX: Federated learning with FaaS at the edge," in *2022 IEEE 18th International Conference on e-Science (e-Science)*, pp. 11–20, IEEE, 2022.

[27] J. G. Pauloski, Y. Babuji, R. Chard, M. Sakarvadia, K. Chard, and I. Foster, "Empowering Scientific Workflows with Federated Agents," 2025.

[28] T.-H. Hoang, J. Fuhrman, R. Madduri, M. Li, P. Chaturvedi, Z. Li, K. Kim, M. Ryu, R. Chard, E. Huerta, *et al.*, "Enabling end-to-end secure federated learning in biomedical research on heterogeneous computing environments with appflx," *arXiv preprint arXiv:2312.08701*, 2023.

[29] P. Patros, M. Ooi, V. Huang, M. Mayo, C. Anderson, S. Burroughs, M. Baughman, O. Almurshed, O. Rana, R. Chard, K. Chard, and I. Foster, "Rural AI: Serverless-powered federated learning for remote applications," *IEEE Internet Computing*, vol. 27, no. 2, pp. 28–34, 2023.

[30] H. Devaraj, S. Sohail, M. Ooi, B. Li, N. Hudson, M. Baughman, K. Chard, R. Chard, E. Casella, I. Foster, and O. Rana, "RuralAI in tomato farming: Integrated sensor system, distributed computing, and hierarchical federated learning for crop health monitoring," *IEEE Sensors Letters*, vol. 8, no. 5, pp. 1–4, 2024.

[31] M. P.-L. Ooi, S. Sohail, V. G. Huang, N. Hudson, M. Baughman, O. Rana, A. Hinze, K. Chard, R. Chard, I. Foster, T. Spyridopoulos, and H. Nagra, "Measurement and applications: Exploring the challenges and opportunities of hierarchical federated learning in sensor applications," *IEEE Instrumentation & Measurement Magazine*, vol. 26, no. 9, pp. 21–31, 2023.

[32] J. G. Pauloski, V. Hayot-Sasson, L. Ward, N. Hudson, C. Sabino, M. Baughman, K. Chard, and I. Foster, "Accelerating Communications in Federated Applications with Transparent Object Proxies," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, (New York, NY, USA), Association for Computing Machinery, 2023.

[33] J. G. Pauloski, V. Hayot-Sasson, L. Ward, A. Brace, A. Bauer, K. Chard, and I. Foster, "Object Proxy Patterns for Accelerating Distributed Applications," 2024.

[34] G. Pauloski, K. Chard, and I. Foster, "Agentic discovery: Closing the loop with cooperative agents," *IEEE Computer*, 2025.

[35] A. Brace, I. Yakushin, H. Ma, A. Trifan, T. Munson, I. Foster, A. Ramanathan, H. Lee, M. Turilli, and S. Jha, "Coupling streaming AI and HPC ensembles to achieve 100–1000× faster biomolecular simulations," in *Int. Parallel and Distributed Processing Symp.*, pp. 806–816, 2022.

[36] X. Yan, N. Hudson, H. Park, D. Grzenda, J. G. Pauloski, M. Schwarting, H. Pan, H. Harb, S. Foreman, C. Knight, T. Gibbs, K. Chard, S. Chaudhuri, E. Tajkhorshid, I. Foster, M. Moosavi, L. Ward, and E. A. Huerta, "MOFA: Discovering Materials for Carbon Capture with a GenAI- and Simulation-Based Workflow," 2025.

[37] L. Ward, G. Sivaraman, J. G. Pauloski, Y. Babuji, R. Chard, N. Dandu, P. C. Redfern, R. S. Assary, K. Chard, L. A. Curtiss, R. Thakur, and I. Foster, "Colmena: Scalable Machine-Learning-Based Steering of Ensemble Simulations for High Performance Computing," in *IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments*, pp. 9–20, 2021.

[38] L. Ward, J. G. Pauloski, V. Hayot-Sasson, R. Chard, Y. Babuji, G. Sivaraman, S. Choudhury, K. Chard, R. Thakur, and I. Foster, "Cloud Services Enable Efficient AI-Guided Simulation Workflows across Heterogeneous Resources," 2023.

[39] J. Stubbs, R. Cardone, M. Packard, A. Jamthe, S. Padhy, S. Terry, J. Looney, J. Meiring, S. Black, M. Dahan, S. Cleveland, and G. Jacobs, "Tapis: An API platform for reproducible, distributed computational research," in *Future of Information and Communication Conference*, pp. 878–900, Springer, 2021.

[40] D. J. Bard, M. R. Day, B. Enders, R. J. Hartman-Baker, J. Riney, C. Snavely, and G. Torok, "Automation for data-driven research with the NERSC superfacility API," in *High Performance Computing* (H. Jagode, H. Anzt, H. Ltaief, and P. Luszczek, eds.), (Cham), pp. 333–345, Springer International Publishing, 2021.

[41] "Existing workflow systems." `https://s.apache.org/existing-workflow-systems`. Accessed June 2025.

[42] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[43] J. Goecks, A. Nekrutenko, J. Taylor, and G. T. team@ galaxyproject. org, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome biology*, vol. 11, pp. 1–13, 2010.

[44] M. Albrecht, P. Donnelly, P. Bui, and D. Thain, "Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids," in *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, SWEET '12, (New York, NY, USA), pp. 1:1–1:13, ACM, 2012.

[45] R. F. da Silva, H. Casanova, K. Chard, I. Altintas, R. M. Badia, B. Balis, T. Coleman, F. Coppens, F. Di Natale, B. Enders, T. Fahringer, R. Filgueira, G. Fursin, D. Garijo, C. Goble, D. Howell, S. Jha, D. S. Katz, D. Laney, U. Leser, M. Malawski, K. Mehta, L. Pottier, J. Ozik, J. L. Peterson, L. Ramakrishnan, S. Soiland-Reyes, D. Thain, and M. Wolf, "A community roadmap for scientific workflows research and development," in *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*, pp. 81–90, 2021.

[46] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.

[47] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn, "Taverna: A tool for building and running workflows of services," *Nucleic Acids Research*, vol. 34, no. suppl_2, pp. W729–W732, 2006.

[48] "Amazon Simple Workflow Service." `https://docs.aws.amazon.com/amazonswf/latest/developerguide/swf-welcome.html`. Accessed June 2025.

[49] "AWS Step Functions Visual workflows for modern applications." `https://aws.amazon.com/step-functions`. Accessed June 2025.

[50] "GitHub Actions." `https://github.com/features/actions/`. Accessed June 2025.

[51] "AWS CodePipeline." `https://aws.amazon.com/codepipeline/`. Accessed June 2025.