Enabling Remote Management of FaaS Endpoints with Globus Compute Multi-User Endpoints

Rachana Ananthakrishnan
rachana@globus.org
University of Chicago and Argonne National Laboratory
Chicago, Illinois, USA

Yadu Babuji
yadunand@uchicago.edu
University of Chicago and Argonne National Laboratory
Chicago, Illinois, USA

Matt Baughman
mbaughman@uchicago.edu
University of Chicago and Argonne National Laboratory
Chicago, Illinois, USA

Josh Bryan
josh@globus.org
University of Chicago and Argonne National Laboratory
Chicago, Illinois, USA

Kyle Chard
chard@uchicago.edu
University of Chicago and Argonne National Laboratory
Chicago, Illinois, USA

Ryan Chard
rchard@anl.gov
University of Chicago and Argonne National Laboratory
Chicago, Illinois, USA

Ben Clifford
benc@hawaga.org.uk
University of Chicago and Argonne National Laboratory
Chicago, Illinois, USA

Ian Foster
foster@anl.gov
University of Chicago and Argonne National Laboratory
Chicago, Illinois, USA

Daniel S. Katz
d.katz@ieee.org
University of Illinois Urbana-Champaign
Champaign, Illinois, USA

Kevin Hunter Kesling
kevin@globus.org
University of Chicago and Argonne National Laboratory
Chicago, Illinois, USA

Chris Janidlo
cjanidlo@globus.org
University of Chicago and Argonne National Laboratory
Chicago, Illinois, USA

Reid Mello
reid@globus.org
University of Chicago and Argonne National Laboratory
Chicago, Illinois, USA

Lei Wang
lei@globus.org
University of Chicago and Argonne National Laboratory
Chicago, Illinois, USA

ABSTRACT

Globus Compute implements a hybrid Function as a Service (FaaS) model in which a single cloud-hosted service is used by users to manage execution of Python functions on user-owned and -managed Globus Compute endpoints deployed on arbitrary compute resources. Here we describe a new multi-user and multi-configuration Globus Compute endpoint. This system, which can be deployed by administrators in a privileged account, enables dynamic creation of *user endpoints* that are forked as new processes in user space. The multi-user endpoint is designed to provide the security interfaces necessary for deployment on large, shared HPC clusters by, for example, restricting user endpoint configurations, enforcing various authorization policies, and via customizable identity-username mapping.

CCS CONCEPTS

• **Computer systems organization → Distributed architectures**; • **Computing methodologies → Distributed computing methodologies**; • **Human-centered computing → Ubiquitous and mobile computing**.

KEYWORDS

Serverless, Globus Compute, Distributed Computing

This work is licensed under a Creative Commons Attribution International 4.0 License.

PEARC '24, July 21–25, 2024, Providence, RI, USA

© 2024 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0419-2/24/07
https://doi.org/10.1145/3626203.3670612

ACM Reference Format:
Rachana Ananthakrishnan, Yadu Babuji, Matt Baughman, Josh Bryan, Kyle Chard, Ryan Chard, Ben Clifford, Ian Foster, Daniel S. Katz, Kevin Hunter Kesling, Chris Janidlo, Reid Mello, and Lei Wang. 2024. Enabling Remote Management of FaaS Endpoints with Globus Compute Multi-User Endpoints. In *Practice and Experience in Advanced Research Computing (PEARC '24), July 21–25, 2024, Providence, RI, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3626203.3670612

1 INTRODUCTION

Globus Compute, formerly known as funcX [2], is a hybrid cloud-edge Function as a Service (FaaS) platform that enables managed execution of functions across a distributed set of computing "endpoints." The hybrid model combines a robust cloud-hosted service as an interface for all user interactions with a user-installable software agent to create *personal* endpoints. These endpoints provide an interface to arbitrary remote computers to enable remote execution of functions.

The Globus Compute software agent, the Python software responsible for creating a *personal* endpoint, was designed to support a single user. The software uses Parsl [1] to provision resources from the host computing system, for example, via batch schedulers (e.g., Slurm, PBS). The agent allows users to install and configure the endpoint for their use. As a user-installed and managed Python process, users must users must first login (e.g., via SSH) to a target system to pip install the Python-based agent and configure the endpoint. Users must subsequently login to the system to manage the configured endpoint, for example to change the scheduler configuration (e.g., allocation, queue name) or the way that resources are provisioned (e.g., number of nodes in a batch job, number of workers deployed on a node). This approach is problematic from several perspectives. For example, from a user's perspective, it is both cumbersome and error-prone, relying on users to login manually to restart endpoints during system outages or modify configurations. From an administrator perspective, there is no control of the user endpoints deployed on their machines or ability to restrict how an endpoint is used.

To address these challenges, we have developed a new multi-user endpoint (MEP) that can be installed and managed by system administrators to support many users on a single system. Installed from native packages and deployed as a privileged user, the MEP dynamically forks user endpoints (UEPs) as Python processes in user space. The MEP uses a flexible identity mapping approach for translating the identity used to authenticate with Globus Compute to the local user account in which the UEP process is started. We adopt the identity mapping approach that is used by the widely deployed Globus Connect Server software.

2 GLOBUS COMPUTE MULTI-USER ENDPOINTS

A multi-User endpoint (MEP) is effectively a manager of user-endpoints (UEPs). In a typical non-MEP paradigm, a user would SSH to a compute resource (e.g., a cluster's login-node), create a Python virtual environment (e.g., virtualenv, pipx, conda), and then install and run "globus-compute-endpoint" in their user-space. By contrast, a MEP is a deployed as a privileged process (e.g., run as root) by an administrator that then manages child processes for other users. Upon receiving a "start endpoint" request from the Globus Compute cloud service, a MEP creates a user-process by *_fork()_&_drop privileges_&_exec()_* pattern, and then watches that child process until it terminates. User tasks are sent directly to the UEP. That is, the MEP does not receive or execute tasks.

```
1       from globus_compute_sdk import Executor
2
3       def some_task(*a, **k):
4           return 1
5
6       mep_endpoint_id = "..."
7       config = { ... }
8       with Executor(endpoint_id=mep_endpoint_id,
         user_endpoint_config=config) as ex:
9           fut = ex.submit(some_task)
10          print("Result:", fut.result())
```

2.1 User workflow

Users interact with a MEP in the same way they currently interact with personal endpoints. After discovering an endpoint ID, for example, via the Globus web service or site-specific documentation, the user invokes a function using the Globus Compute Python SDK (Listing 2). The only difference, is that a user can specify the configuration for the user endpoint (e.g., the queue and allocation to be used, number of workers to deploy on a node, and Conda environment in which to execute tasks). As shown in Figure 1, the workflow proceeds as follows.

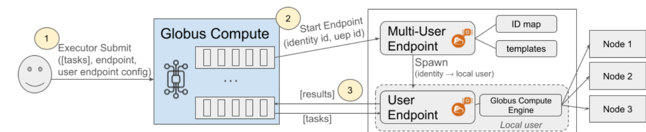


Figure 1: Globus Compute Multi-User workflow. 1) A user submits tasks to be executed, specifying the MEP ID and a configuration of their user endpoint. 2) Globus Compute issues a start endpoint request to the MEP, passing the user identity and the UEP ID to be started. Credentials to access the UEP queue are also supplied. 3) The UEP is started and retrieves tasks from Globus Compute, provisions local resources, executes the tasks, and returns results to Globus Compute. The UEP shuts down after some period of inactivity.

(1) The SDK sends a POST request to the Globus Compute web service.
(2) The web service identifies the endpoint as a MEP and generates a UEP id specific to the tuple of the 'mep_endpoint_id', the identity id of the user making the request, and the endpoint configuration in the request (e.g., "tuple(endpoint_id, identity_id, endpoint_configuration)" this identifier is simultaneously stable and unique.
(3) The web service sends a `start-UEP` request to the MEP (via AMQP), asking it to start an endpoint identified by the id generated in the previous step, and as the user identified by the REST request.
(4) The MEP maps the Globus Auth identity in the `start-UEP` request to a local username.
(5) The MEP starts a UEP as the UID from the previous step.
(6) The UEP communicates with the Globus Compute web service, it accepts the original task submission, executes the task, and returns the results.

2.2 Resource configuration

One benefit of the MEP model is that users can remotely configure UEPs and thus the MEP also serves as a multi-configuration endpoint. Unlike the personal endpoint, which requires that users preconfigure the endpoint before it is started and cannot change that configuration remotely. The UEP configuration model allows users to specify configurable parameters (e.g., allocation, queue name, number of nodes) following a template set by the MEP administrator (`user_config_template.yaml`). For example, in Listing 2.2 we see the user is able to specify their allocation id, queue name, max number of workers per node, and conda environment to load as a JSON document passed when creating the executor. The MEP will reuse the user's existing UEP if executors are constructed with the same endpoint configuration. It determines equivalence by hashing the JSON document and thus users can force new UEPs to be created by modifying any attribute of the configuration (e.g., the label).

```
1 with Executor(
2     endpoint_id=mep_site_id,
3     user_endpoint_config={
4         "account_id": "<ALLOCATION ID>",
5         "queue": "gpu",
6         "max_workers_per_node": 64,
7         "worker_init": "conda activate environment"}
8 ) as ex:
9     fut = ex.submit(some_task)
10    print("Result:", fut.result())
```

2.3 Identity mapping

Before forking the UEP process, the MEP must map the identity used to authenticate with Globus Compute (a Globus Auth Identity Set) to a local username on the MEP resource. The local username is passed to `getpwnam(3)` to ascertain a UID for the user and then to fork the UEP process. Rather than develop a new mapping method, we instead use the flexible method used by Globus Connect Server. We allow administrators to specify a mapping file in the MEP `config.yaml` file. The mapping file supports various methods for converting an identity set to a local user account including static mappings (e.g., bob@uchicago.edu -> bob), rule-based mappings based on identity provider (e.g., map <username>@uchicago.edu to <username>), and even using an external program. Note: if an administrator has previously configured a mapping file with Globus Connect Server, they can reuse the same mapping file for the Globus Compute MEP.

2.4 Restricting user configuration

Discussions with administrators indicated the need to restrict user configurations to, for example, better support use of their systems (e.g., setting the scheduler type, listing known queue names, predefining sbatch arguments necessary to mount file systems, and specifying internal network interfaces for communication) and ensure use follows defined policies (e.g., node counts, wall times). We support these restrictions by enabling MEP administrators to define *templates* that specify set properties while leaving others as variables that can be specified in the user endpoint configuration. Further, we allowed administrators to also restrict the range of functions that can be executed on an endpoint by specifying a whitelist of function IDs. Finally, as part of the identity mapping, administrators can restrict which users are able to access the endpoint.

2.5 Security model

The MEP implements a comprehensive and multi-layer security model to ensure only authorized execution of tasks. Globus Compute uses Globus Auth to manage user authentication and authorization. All requests to Globus Compute are associated with an user identity set. The identity set can include one or more linked identities from the thousands of supported identity providers (e.g., InCommon, Google, ORCID). As described above, we use the same identity mapping approach that is used by Globus Connect Server to determine to which user account to map an identity set. This model has been validated by security reviews and is in production in more than 30,000 Globus Connect Server deployments.

Globus Auth supports the concept of Auth *Policies*, which govern authentication timeouts and allowed domains from which a user may use linked identities to access a MEP endpoint. Auth Policies can be specified in the MEP configuration file. For example, if deployed on a supercomputer at the Argonne Leadership Computing Facility (ALCF), the authentication domain can be set to "alcf.anl.gov," and authentication timeout can be set to 12 hours. The user then has to authenticate with an ALCF identity every 12 hours to submit new tasks to the endpoint.

Since MEP adopts a model where all task execution and job submission is done in a process running as a local user, we rely on standard POSIX security controls and privileges to govern what tasks submitted to the endpoint are allowed to do. This means any limitations on resources such as CPU, node allocations, or scheduler queue access that apply to that user's local account will apply to tasks submitted to the Globus Compute UEP on their behalf.

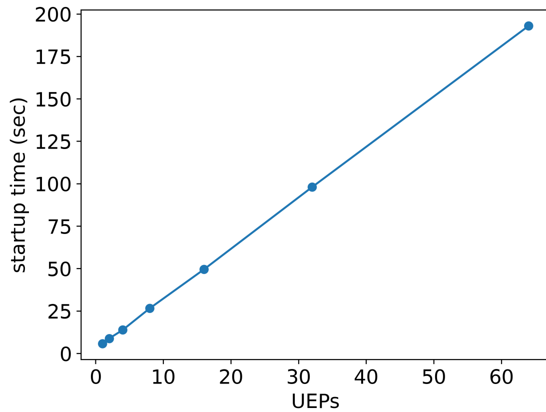
Finally, all communication between Globus Compute and the endpoints is secured to ensure the confidentiality and authenticity of data. Transport Layer Security (TLS) is used to encrypt data in transit. For example, from client to web service, web service to endpoints (over AMQPS), and from endpoints to workers.

3 EVALUATION

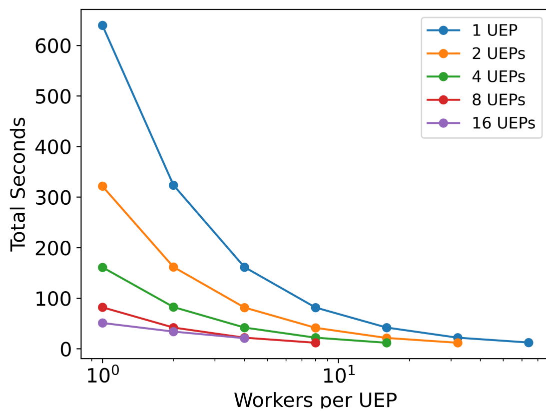
We evaluate the overhead of creating UEPs and the scalability of the MEP deployed on a single node. We run experiments on the Jetstream cloud [7] and deploy the MEP on a m3.2xl instance with 64 CPU cores, 250 GB RAM, and 60 GB root disk. In both experiments, we create a Globus Compute Executor deployed on a laptop and we update that executor with unique UEP configurations by setting the description to a random string.

Figure 2a shows the overhead of deploying UEPs as we increase the number of UEPs from 1 to 64. We record the time from when we submit a "no-op" task to each UEP, until all tasks return. We see creation time is linear as we use a single executor that waits for the UEP to start before starting the next.

Figure 2b shows a strong scaling study where we execute 64 ten-second tasks. We increase the number of UEPs and the number of workers per UEP. We divide the available cores among the UEPs so as not to exceed the capacity of the node. We record the time from when the task is submitted until all results are returned and



(a) Time to create UEP endpoints as we increase the number of UEPs.



(b) Strong scaling experiment to run 64 tasks as we increase the number of UEPs.

Figure 2: Scaling of MEP and UEP endpoints.

therefore include the UEP startup cost. We see good scaling as we increase the number of workers and as we increase the number of UEPs.

4 RELATED WORK

Various abstraction layers have been developed to hide the differences between batch schedulers. For example, SAGA [6], DRMAA [11] and PSI/J [8] provide common interfaces for submitting batch jobs to schedulers. Systems, such as Globus Toolkit [5] and Unicore [10] provided remote interfaces for job submission in Grid Computing environments. For example, Globus GRAM [3] implemented various services for submitting and monitoring jobs. GRAM was administrator deployed and exposed a web service API for job submission. It used Grid Security methods for authentication/authorization and mapped users to local accounts using a gridmap file. Unlike, Globus Compute, it supported only batch jobs rather than programming functions and did not provide the flexible configuration and restrictions offered by Globus Compute MEPs. Workflow systems, such as Parsl [1] and Pegasus [4] offer remote computing capabilities. These systems coordinate execution as a single user and rely on methods like pre-established SSH connections for remote execution. Open OnDemand [9] provides a web-based interface to remote HPC computers. Users can login via their institutional identity, manage data, and create and manage batch jobs. Open OnDemand provides a template for submitting batch jobs, mirroring the underlying scheduler batch file format. Open OnDemand is installed as a web application in Apache on the target system and relies on supported Apache authentication. Identity mapping is performed using a similar method to Globus Compute by using static mappings, regex patterns, and via scripts. Globus Compute provides a higher level interface for function execution across connected endpoints.

5 SUMMARY

Globus Compute multi-user endpoints enable administrator-managed deployment of FaaS capabilities on existing computing resources, from clouds to HPC clusters. The multi-user endpoint is able to fork single-user endpoints in local user accounts using the widely-deployed Globus identity mapping model. Users can dynamically configure an endpoint, via the cloud-hosted Globus Compute API, specifying, for example, allocation name, queue name, and resources to be allocated. Importantly, administrators can restrict these endpoint configurations and provide templated options to users. Administrators also have complete control of which users are permitted to access an endpoint and what local user accounts those users are to be mapped to.

ACKNOWLEDGMENTS

This work was supported in part by NSF 2004894/2004932 and Laboratory Directed Research and Development funding from Argonne National Laboratory under U.S. Department of Energy under Contract DE-AC02-06CH11357 and used resources of the Argonne Leadership Computing Facility.

REFERENCES

[1] Yadu Babuji, Anna Woodard, Zhuozhao Li, Daniel S. Katz, Ben Clifford, Rohan Kumar, Lukasz Lacinski, Ryan Chard, Justin Wozniak, Ian Foster, Mike Wilde, and Kyle Chard. 2019. Parsl: Pervasive Parallel Programming in Python. In *28th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*.

[2] Ryan Chard, Yadu Babuji, Zhuozhao Li, Tyler Skluzacek, Anna Woodard, Ben Blaiszik, Ian Foster, and Kyle Chard. 2020. funcX: A Federated Function Serving Fabric for Science. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing* (Stockholm, Sweden) *(HPDC '20)*. Association for Computing Machinery, New York, NY, USA, 65–76. https://doi.org/10.1145/3369583.3392683

[3] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. 1998. A resource management architecture for metacomputing systems. In *Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson and Larry Rudolph (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 62–82.

[4] Ewa Deelman, Karan Vahi, Mats Rynge, Rajiv Mayani, Rafael Ferreira da Silva, George Papadimitriou, and Miron Livny. 2019. The Evolution of the Pegasus Workflow Management Software. *Computing in Science & Engineering* 21, 4 (2019), 22–36. https://doi.org/10.1109/MCSE.2019.2919690

[5] Ian Foster and Carl Kesselman. 1998. The Globus Toolkit. In *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 259–278.

[6] Tom Goodale, Shantenu Jha, Hartmut Kaiser, Thilo Kielmann, Pascal Kleijer, Gregor von Laszewski, Craig Lee, Andre Merzky, Hrabri Rajic, and John Shalf. 2006. SAGA: A Simple API for Grid Applications. High-level application programming on the Grid. *Computational Methods in Science and Technology* 12, 1 (2006), 7–20.

[7] David Y. Hancock, Jeremy Fischer, John Michael Lowe, Winona Snapp-Childs, Marlon Pierce, Suresh Marru, J. Eric Coulter, Matthew Vaughn, Brian Beck, Nirav Merchant, Edwin Skidmore, and Gwen Jacobs. 2021. Jetstream2: Accelerating cloud computing via Jetstream. In *Practice and Experience in Advanced Research Computing* (Boston, MA, USA) *(PEARC '21)*. Article 11, 8 pages. https://doi.org/10.1145/3437359.3465565

[8] Mihael Hategan-Marandiuc, Andre Merzky, Nicholson Collier, Ketan Maheshwari, Jonathan Ozik, Matteo Turilli, Andreas Wilke, Justin M. Wozniak, Kyle Chard, Ian Foster, Rafael Ferreira da Silva, Shantenu Jha, and Daniel Laney. 2023. PSI/J: A Portable Interface for Submitting, Monitoring, and Managing Jobs. In *2023 IEEE 19th International Conference on e-Science (e-Science)*. 1–10. https://doi.org/10.1109/e-Science58273.2023.10254912

[9] Dave Hudak, Doug Johnson, Alan Chalker, Jeremy Nicklas, Eric Franz, Trey Dockendorf, and Brian L. McMichael. 2018. Open OnDemand: A web-based client portal for HPC centers. *Journal of Open Source Software* 3, 25 (2018), 622.

[10] Mathilde Romberg. 2002. The UNICORE grid infrastructure. *Scientific Programming* 10, 2 (2002), 149–157.

[11] Peter Troger, Hrabri Rajic, Andreas Haas, and Piotr Domagalski. 2007. Standardization of an API for Distributed Resource Management Systems. In *7th IEEE International Symposium on Cluster Computing and the Grid*. 619–626.