Globus Timers: Scheduling Periodic Data Management Actions on Distributed Research Infrastructure

Rachana Ananthakrishnan
rachana@globus.org
University of Chicago
Chicago, Illinois, USA

Josh Bryan
josh@globus.org
University of Chicago
Chicago, Illinois, USA

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

Ryan Chard
rchard@anl.gov
Argonne National Laboratory
Lemont, Illinois, USA

Kurt McKee
kurt@globus.org
University of Chicago
Chicago, Illinois, USA

Ada Nikolaidis
ada@globus.org
University of Chicago
Chicago, Illinois, USA

Jim Pruyne
pruyne@globus.org
University of Chicago
Chicago, Illinois, USA

Stephen Rosen
sirosen@globus.org
University of Chicago
Chicago, Illinois, USA

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

ABSTRACT

The increasing complexity and scale of scientific problems presents new challenges to manage, analyze, and manipulate large volumes of data. Automation is critical to reducing the time spent on mundane and error-prone tasks, such as backing up data, purging old records, and even performing routine analysis. Globus provides a broad range of research automation services to empower users to outsource and automate data-oriented tasks, such as transfer, publication, and analysis. Here we present the newest addition to the Globus platform, the Globus Timers service. Timers is a cron-like service designed to simplify scheduling repeated actions. It leverages the Globus Auth service to engage securely any service exposing a compatible interface. Since its release in late 2021, Timers has been used by over 1300 users to perform more than 2.2 million tasks. We explain how Timers can be used, describe its implementation, and discuss its adoption.

ACM Reference Format:
Rachana Ananthakrishnan, Josh Bryan, Kyle Chard, Ryan Chard, Kurt McKee, Ada Nikolaidis, Jim Pruyne, Stephen Rosen, and Ian Foster. 2023. Globus Timers: Scheduling Periodic Data Management Actions on Distributed Research Infrastructure. In *Practice and Experience in Advanced Research Computing (PEARC '23), July 23–27, 2023, Portland, OR, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3569951.3597571

PEARC '23, July 23–27, 2023, Portland, OR, USA

ACM ISBN 978-1-4503-9985-2/23/07…$15.00
https://doi.org/10.1145/3569951.3597571

1 INTRODUCTION

The growing volume and complexity of data produced by advanced research methods necessitates the use of automated systems for efficient management and use of data. Automation can both reduce the risk of human error and reduce the time spent on labor-intensive tasks, allowing scientists to focus on generating insights from their data. Further, automation can foster reproducibility, which is essential for effective collaboration and the validation of findings.

Globus [3] provides a suite of services for research data management, including data transfer, sharing, and publication. These cloud-hosted services enable researchers to manage data across institutions and computational and storage resources. For example, Globus Transfer allows users to transfer data between various storage systems, automating movement while handling potential errors or interruptions.

As Globus becomes an ever more critical tool for researchers, there is a growing need perform tasks without requiring human intervention: e.g., to backup datasets to tape storage; replicate and synchronize data across geographically distributed computers used for analysis tasks; and remove unused data after some period of time. Thus, there is a need for cron-like behavior via which users can schedule tasks to be performed at predefined intervals or specific times.

Recently, we introduced Globus Automation Services [1] to empower researchers to define and run data management flows that span resources and time scales. Here we present the newest addition to the suite of Globus Automation Services: the Globus Timers service. Timers is a cloud-hosted platform that offers the ability to schedule and automate various data management tasks. For example, users can define timers that periodically synchronize data movement between across storage systems or schedule flows to analyze data at specific times. This scheduling capability is designed

to help researchers reliably outsource and automate routine data management tasks.

The remainder of this paper is organized as follows. §2 discusses how Globus Timers can be used. §3 describes the architecture and implementation of the Timers service. §4 reviews the adoption and usage of Timers, and §5 and §6 present related work and conclusions, respectively.

2 USING TIMERS

The Timers service enables users to schedule, either as a one-off or recurring, arbitrary "tasks" (often via other Globus services). User's can securely delegate permission to a `timer` to perform actions on their behalf, such as performing a transfer or initiating analysis jobs. Here we describe how users work with the Timers service.

Timers exposes three primary interfaces: the SDK, CLI, and Web App. To use a timer one must first *register* the `timer` with the service. When registering a `timer`, the user specifies a schedule and the *action* to be performed. Currently the Globus Web App only supports registration of `timers` for Globus Transfers. `Timers` that invoke other actions must be specified via the SDK or CLI.

The following schema is supported when defining a new timer:

```
{
    "name": "string",
    "stop_after": { "date": "2023-08-24T14:15:22Z",
                    "n_runs": int },
    "interval": int,
    "scope": "string",
    "refresh_token": "string",
    "callback_url": [ "string" ],
    "callback_body": { "request_id": "uuid",
                       "body": {} },
    "start": "20123-08-24T14:15:22Z"
}
```

The submission schema requires several pieces of information, including the action, interval, token, and stopping condition. The action is defined by the *callback_url* and *callback_body*. These fields specify which service the `timer` will invoke when "triggered" and the payload to pass to the action. For example, the above listing specifies a Transfer action and would be accompanied by a payload with the source and destination endpoint for which to perform a transfer. A *refresh_token* is used to grant the timer permission to perform the action on the user's behalf. The *start* and *interval* fields specify when the timer will begin operating and the rate at which it will be triggered. The *stop_after* field specifies the exit condition for the timer. This may either be a specific time after which the timer will be invalidated, or a number of runs (*n_runs*) to be performed by the timer. A timer can also be created via the Globus CLI, for example:

```
globus timer create transfer --interval 1d -r $e1:/ $e2:/
```

Once a `timer` is deployed it can be *listed* and managed through any of the available interfaces. The listing provides information relating to the `timer's` status and configuration, as well as details and results regarding previous runs, most recent run, and when the next run is scheduled to occur. Management operations include *pause*, *resume*, and *delete*. When a `timer` is in the *deployed* state, the Timers service manages the reliable invocation of the `timer's` specified action at the appropriate time and continues to schedule the `timer` until the completion criteria is met.

In addition to registering a `timer` with the Timers service through the SDK or CLI, we support use of `timers` with the Transfer service through the Globus Web App. The web interface to deploy a `timer` is shown in Fig. 1. This interface provides a simple mechanism to schedule transfers using a graphical interface to select the rate and completion criteria.

`Timers` can be managed through the Web App, as shown in Fig. 2. This interface allows users to monitor `timers` and quickly inspect their progress. Users can also use this interface to pause, resume, or delete `timers` at their discretion.

3 GLOBUS TIMERS SERVICE

As shown in Fig. 3, the Globus Timers architecture comprises three key components: a scheduler, task queue, and workers. We describe the design of these components and their interactions and then present implementation details.

The **scheduler** is responsible for evaluating `timers` and routing actions for execution as needed. The scheduler operates in a loop, continuously surveying the set of deployed `timers` to determine whether any have triggered (i.e., met their scheduled time for execution). Once a `timer` is identified as needing to be executed, the scheduler places the `timer` onto the task queue. Once placed on the task queue, the scheduler then computes the next scheduled time for the `timer` to trigger and updates the job status. **Task queues** separate the scheduling and management of `timers` from their execution. A pool of **workers** monitor the task queue and, when a task is available, remove it from the queue for execution. The worker then invokes the action specified in the callback address with the associated action payload.

These three components provide a reliable and scalable means to abstract the processes monitoring `timers` from those that perform their associated actions. The Timers service design is inherently scalable, allowing both schedulers and workers to be horizontally scaled to meet demand. This approach provides both the robustness necessary to fulfill millions of `timers`, as well as the reliability necessitated by a cron-like system.

Implementation: The Timer service is implemented using FastAPI to provide an asynchronous and performant REST interface. The Timer API is described below.

- `POST <timer_url>/jobs/`: Register a new `timer`.
- `GET <timer_url>/jobs/`: List your `timers`.
- `GET <timer_url>/jobs/<job_id>/`: Get the information and status of a `timer`.
- `DELETE <timer_url>/jobs/<job_id>/`: Delete a `timer`.
- `PATCH <timer_url>/jobs/<job_id>/`: Update a `timer`.
- `POST <timer_url>/jobs/<job_id>/pause/`: Pause a `timer`.
- `POST <timer_url>/jobs/<job_id>/resume/`: Resume a paused `timer`.

`timers` are stored in a Amazon Relational Database Service (RDS). Triggered `timers` are placed in the task queue for execution. The task queue is implemented as a managed AWS ElastiCache Redis cluster. The `timer` includes the action callback address, authentication token to communicate with the action provider, and the payload to pass when executing the action. Tasks are processed in first in, first out (FIFO) order.

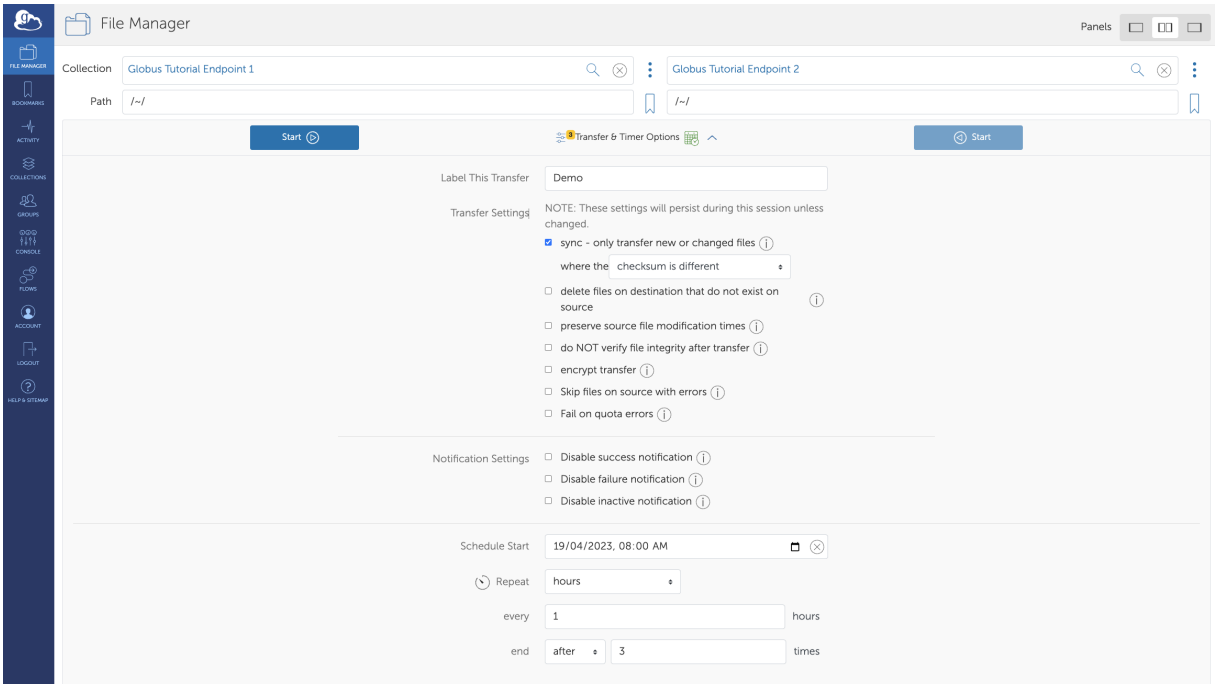

Figure 1: The Web App interface to configure and deploy a `timer` to schedule the use of Globus Transfer. In addition to source and destination information, users can specify the rate and duration of the `timer`.

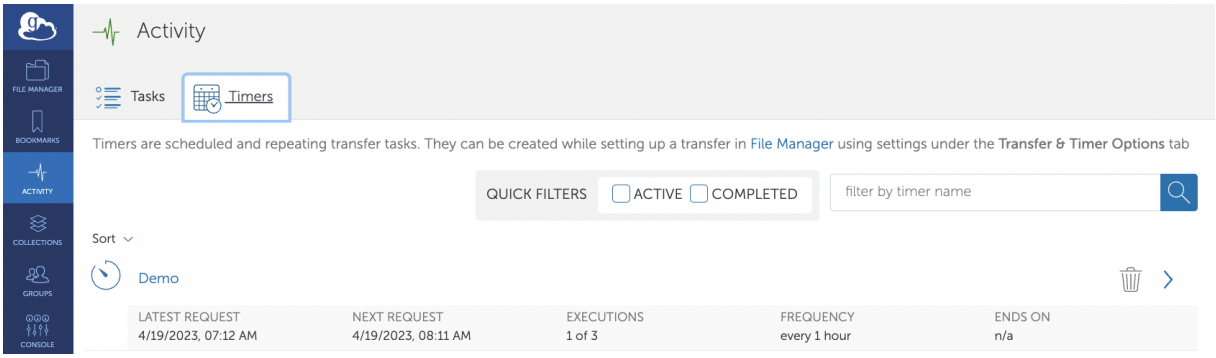

Figure 2: The Timer activity pane of the Globus Web App. This page allows users to inspect and manage their deployed `timers`. Instances of the `timer`'s execution are visible in the Tasks tab.

The Timers service uses a pool of workers to process tasks from the task queue. Workers are implemented as processes running in containers. Once a worker retrieves a task from the queue it invokes the specified action, via the Globus Action Provider interface [1], using the authentication token to act on the user's behalf. The Action Provider interface [1] provides a common way of asynchronously invoking external actions and is used in Globus flows to create workflows that orchestrate arbitrary actions. `timers` can be used to invoke any service exposing an action provider interface, including more than one dozen Globus services.

Deployment: The Timers service is deployed as a set of containers on the Amazon Elastic Container Service (ECS) alongside a managed ElastiCache Redis cluster and RDS database. AWS provides numerous benefits to hosting the Timers service, such as increased efficiency, scalability, reliability, and monitoring. ECS, as a managed container orchestration service, simplifies the deployment, management, and scaling of Timers containers and enables the service to leverage elasticity, high availability, and robust infrastructure. This combination simplifies the scaling of the Timers service by allowing the horizontal deployment of additional containers as needed.

Authentication and Authorization: The Timers service uses Globus Auth [5] for identity and access management. Thus users

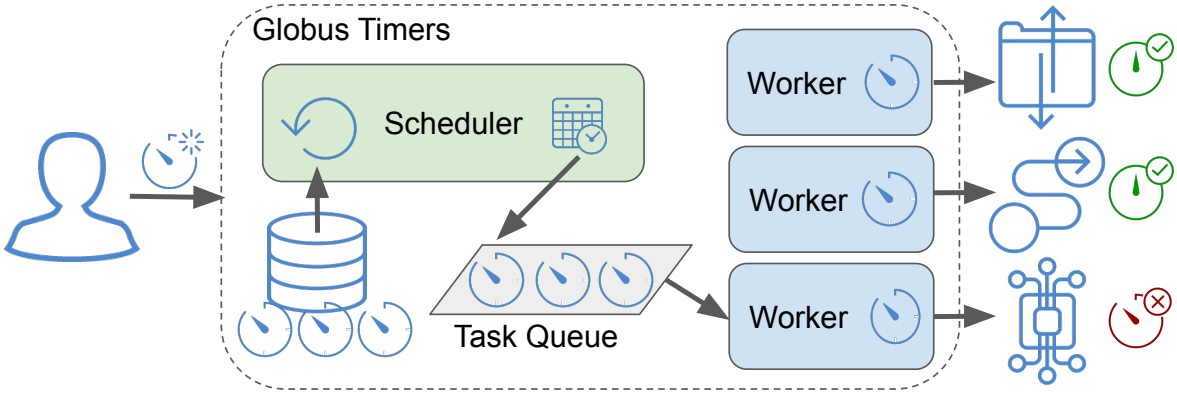

Figure 3: Architecture of the Globus Timers service. Users can register new timers which are stored and then processed by the scheduler. When ready to execute, the scheduler places the timer on the task queue for a worker to process and invoke the specified action.

can log in to Timers using any one of thousands of supported identity providers, such as their university or research institution, and then use those credentials to access resources and services without having to log in again.

Globus Auth exposes industry-standard protocols such as OpenID Connect and OAuth2 to interoperate with a wide range of applications and services. Timers leverages these protocols to enable secure invocation of external actions. Thus, users can consent to allow a `timer` to invoke an external action (e.g., Transfer) as the user. Each `timer` is registered with a refresh token with the appropriate scopes to perform the specified action.

4 ADOPTION

Timers has been used by 1396 users since late 2021 to register 6618 `timers` that have collectively triggered over 2.2 million times. 616 `timers` are currently deployed, 2497 have completed after performing their function and 3505 have been deleted. Fig. 4 shows the total number of `timer` tasks and failures per month since the beginning of 2022. There have been a total of 2 215 022 `timer` tasks, with 2 202 038 successes and 12 984 failures. Failures are typically a result of either invalid input or insufficient permissions for the `timer` to execute the targeted action. Failures are reported by the worker and can be viewed and inspected through the Web App or by querying the status of a `timer's` activities.

Fig. 5 shows the distribution of registered intervals for both all (top) and currently deployed (bottom) `timers`. A *None* interval denotes timers that were scheduled to perform a one-off execution, of which there have been 2449. The figure shows that, of repeating `timers`, 1329 (and 272 currently deployed `timers`) have a scheduled interval between one day and one week. This is due to many use cases performing either daily or weekly synchronization and backup tasks. 101 `timers` were scheduled with an interval greater than 30 days. We suspect these tasks are intended to be performed once each month, but require additional analysis to understand the use cases in which longer intervals are being used.

5 RELATED WORK

Cron, a time-based job scheduler available on Unix-like systems, is widely used to automate tasks that need to be executed at specific times or intervals. Users schedule tasks, such as running scripts or commands, by specifying, in a configuration file called the *crontab*, fields that define the task's timing requirements and the command to invoke when triggered. These concepts have also been applied to distributed systems, where a cron system is established and synchronized to perform time-based job scheduling globally [2].

CloudWatch Events [4], part of Amazon EventBridge, supports cron and rate expressions to schedule operations. Users can define cron rules in Events to trigger actions at a specified time on a certain day of each week or month. Rate expressions allow users to define rules that trigger at a regular rate, such as once every hour or once every day. These rules can be applied to engage other Amazon services, such as invoking a Lambda function or processing logs.

Like these systems, Globus Timers enables users to define time-based schedules for jobs; it differs by being tightly integrated with the Globus fabric, which allows users to automate use of Globus services to orchestrate and perform data management tasks.

6 CONCLUSIONS

We described the latest addition to the Globus platform: the Globus Timers service. Timers is a cron-like system designed to allow users to schedule time-based workloads and configure recurring actions, such as performing regular data backups. Timers can securely interact with any service that exposes an Action Provider interface, which includes most Globus services. Since its launch in late 2021, 1396 users have used the Timers service to carry out more than 2.2 million tasks. In future work we aim to investigate common usage patterns for Timers and develop additional integration's that simplify the deployment and configuration of complex schedules.

ACKNOWLEDGMENTS

This work was supported in part by NSF grant OAC-1835890 and by U.S. DOE contract DE-AC02-06CH11357. We thank Rudyard Richter for his work to pilot the service.

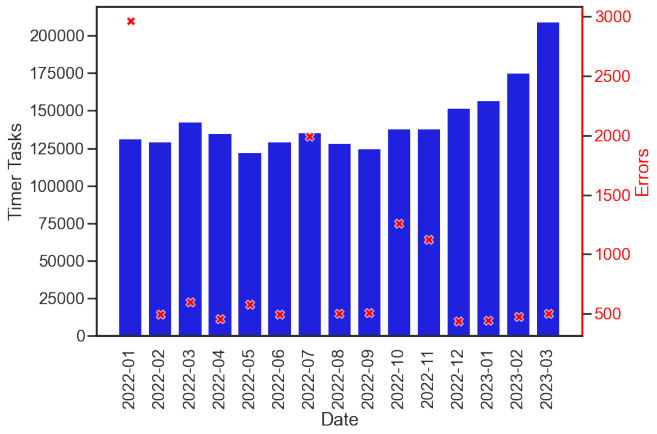

Figure 4: Total `timer` tasks and failures. The blue bars show the total number of `timer` tasks each month since Jan 2021. The number of failures each month are shown as red crosses.

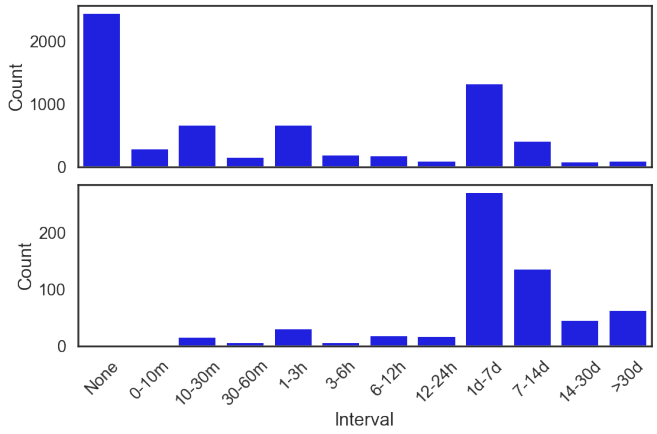

Figure 5: Number of `timers` per repeat interval (or None, meaning no repeated schedule): all registered `timers` (top) and currently deployed `timers` as of April, 2023 (bottom).

REFERENCES

[1] R Chard et al. 2023. Globus automation services: Research process automation across the space–time continuum. *Future Gen. Computer Sys.* (2023).
[2] S Davidovi and K Guliani. 2015. Reliable Cron across the Planet: ... or How I stopped worrying and learned to love time. *Queue* 13, 3 (2015), 30–39.
[3] I Foster. 2011. Globus Online: Accelerating and democratizing science through cloud-based services. *IEEE Internet Computing* 15, 3 (2011), 70.
[4] Amazon Web Services. 2023. *CloudWatch Events.* Retrieved April 2023 from https://bit.ly/3N283M3
[5] S Tuecke et al. 2016. Globus Auth: A research identity and access management platform. In *IEEE 12th International Conference on e-Science.* 203–212.