Google Summer of Code 2019 Proposal

CERN-HSF - Rucio

Pujan Mehta

1. Basic Details

Full Name	Pujan Rajesh Mehta
Institute	A 3rd Year B.E. Computer Engineering student at D.J. Sanghvi College of Engineering, Mumbai.
Email	pujanrmehta@gmail.com
Phone	+91-9930437371
GitHub	github.com/pujanm
LinkedIn	linkedin.com/in/pujanm/
Location & Time Zone	Mumbai, India. Indian Standard Time (GMT +0530)
Address	2/B Manisha Building, Bhardawadi Road, opp. Oscar Diamond Building, Bhardawadi Road, Andheri West, Mumbai - 400058.
Resume	Pujan Mehta Resume

2. About Me

I am Pujan Mehta, a 3rd-year Computer Engineering student at D. J. Sanghvi College of Engineering. I have fluency in programming languages like Python, Java, and C. I am also a full-stack Web developer.

Most of the projects that I have worked on until now are based on Python and I also started my programming career in 2016 with Python with which I realized the true power of Python and also found that many of the big organizations around the world use it heavily. Having knowledge in Software Development helped me get an internship at a

Silicon-Valley based startup named Falkonry where I used Python, Java, Node.js, and Git and learned Docker, Kubernetes, and ELK Stack and also got to work on AWS. Recently, I also completed MOOCs on Machine Learning and Deep Learning during which I used tools like NumPy, Pandas, Matplotlib, Scikit-Learn, and PyTorch and also got an internship at VoyaGenius Labs. My background knowledge comprises of Algorithms, Operating Systems, and Machine Learning.

And apart from these, I am also a student mentor at DJ Unicode a college level open-source organization where I guide juniors on their back-end development projects and I also like Competitive Programming and I take part in competitions held on Codechef and HackerRank.

Why GSOC with CERN-HSF?

I was always fascinated by the work going at CERN and the heavy use of computing along with physics. I also wanted to be associated with CERN in some or the other way and I found GSOC the best way to achieve it and here I will also get a chance to enhance my practical skills as there are varied types of tasks each one of them requiring a set of different concepts. And along with Rucio, I would get a chance to enhance its core features which helps CERN in data management for its large-scale scientific projects easily.

How much time will I be able to commit to the project?

I will be able to contribute 6-8 hours per day to the project. I will be able to work from 9.00 AM (Central European Timezone) till 5.00 PM (Central European Timezone) and I will be working full-time only for GSOC throughout the summer, as I have no other work commitments.

3. Synopsis

Rucio is a data management system for large-scale scientific experiments and is highly scalable and flexible and is used in the ATLAS experiment for large amounts of data transfers and for other processing and due to its heavy use other ongoing global experiments are also trying to adapt Rucio. In which the US HPC centers require GlobusOnline as a transfer option from/to the site which is not supported by Rucio currently.
My task will be to design and implement a transfer tool based on GlobusOnline which supports Rucio internally and allows interactions with Rucio in order to make the transfers successful which will be an added feature to Rucio ultimately.
The other task will be to track account of usage and interests per DID and account.

4. Project Goals

Objectives

- Setup a Rucio development environment either Docker or UNIX based.
- Design for the GlobusOnline transfertool.
- Implementation of the GlobusOnline transfertool interface which will have methods similar to this.
- Add GlobusOnline functionality for submitting and querying transfers inside **rucio.core.transfer** and **rucio.core.request**.
- Integrating the GlobusOnline transfertool with Rucio Conveyor submitter and testing using provided endpoints.
- To also work on this issue.

Tasks

1. **Design of the Globus based transfer tool (features to be covered):**

1) Access & Auth Token Generation (Delegate Proxy)

Input Params:

Registered app **CLIENT_ID** and **CLIENT_SECRET**.

Globus APIs to be called:

ConfidentialAppAuthClient (client),
client.oauth2_client_credentials_tokens

Returns:

auth_access_token and **transfer_access_token**

2) Submit Transfer (Bulk Transfer)

Input Params:

transfer_access_token,
source_endpoint_ids, destination_endpoint_ids,
source and **destination file_paths** or **directory_paths**.

Globus APIs to be called:

AccessTokenAuthorizer, TransferClient,
TransferData, TransferClient.submit_transfer,

TransferData.add_item

Returns:

A list of unique **task_ids.**

3) Cancel Transfer

Input Params:

transfer_access_token, **transfer_id.**

Globus APIs to be called:

AccessTokenAuthorizer, TransferClient,
TransferClient.cancel_task

Returns:

A **message.**

4) Get Task Details (Bulk Query)

Input Params:

transfer_access_token, **transfer_ids list.**

Globus APIs to be called:

AccessTokenAuthorizer, TransferClient,
TransferClient.get_task

Returns:

A dictionary with the all the **tasks** related fields**.**

2. Implementation flow:

In the implementation, the first part will be to develop an abstract class named **GOTransferTool** based on the base abstract class provided here and adding additional methods to the class as described in the design of the transfer tool and other helper methods as required on the go. Then after building the interface I will test it locally by calling the different methods built and test their working.

Then we will have to setup the **GO** authentication where we will have to first add **CLIENT_ID** and **CLIENT_SECRET** inside **rucio.cfg** file. Then when we call the **delegate_proxy** method we will use these client credentials and we will receive a **transfer_token** which we can again store it inside the **rucio.cfg** file as we will have to use this token further for transfers and other querying options.
Then based based based on the status of the transfer **ACTIVE/COMPLETED/**

FAILED we **auto-update/refresh** the token automatically without involvement of the user as we will already saved the client credentials inside the **rucio.cfg** file.

Further, will call the **GOTransferTool** class inside the file **rucio.core.transfer** which is for handling all the transfers to add **GlobusOnline** transfer option inside the **submit_transfer** method and **bulk_query_transfers** method and also inside the file **rucio.core.request** which is for handling various requests to add **GlobusOnline** as an option inside **query_request, query_request_details, cancel_request_did,** and **cancel_request_external_id** methods.
Then whenever a new RSE is added and it supports only **GlobusOnline** as a transfer protocol then the first step will be to add a new attribute to the RSE which can be done as shown below:

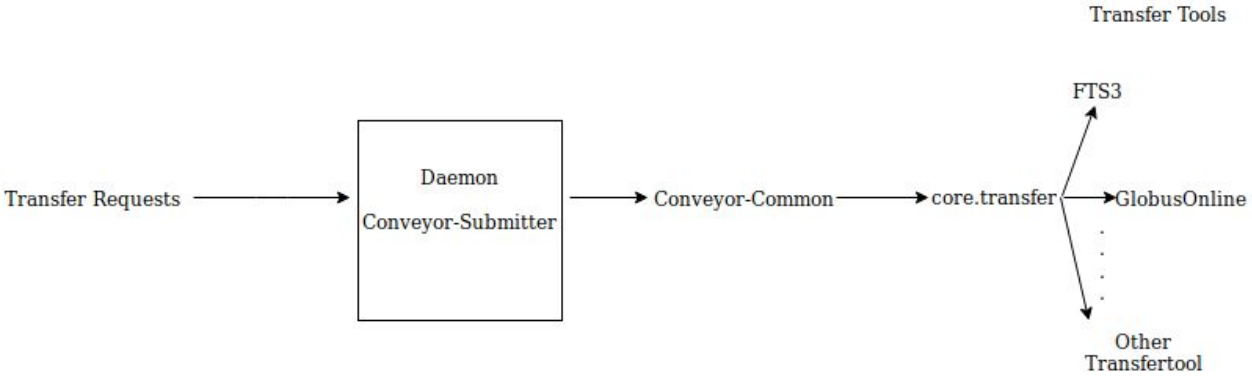
```
[root@7128ad41f038 rucio]# rucio-admin rse set-attribute --rse SITE1_DISK --key transfertool --value globus
Added new RSE attribute for SITE1_DISK: transfertool-globus
[root@7128ad41f038 rucio]# rucio list-rse-attributes SITE1_DISK
SITE1_DISK:    True
istape:        False
transfertool:  globus
[root@7128ad41f038 rucio]#
```

This has to be done so that further when a new transfer request comes to the **rucio-conveyor-submitter** we can divert the request to the correct transfertool. Inside the **conveyor-submitter** daemon we will check the RSE attribute **transfertool** whether it is **globus** or not and pass the parameter **transfertool** inside the **submit_transfer** method accordingly, if there is not such attribute given to the RSE we can assume it supports **FTS3** as the transfertool.

Then inside **conveyor-common** we will have the **submit_transfer** method which is passed from the **conveyor-submitter** we will have to perform a check to find out the correct **transfertool** to pass on to the **submit_transfer** which is inside the **core.transfer.** The code inside **conveyor-common** looks as follows:

```
eid = transfer_core.submit_bulk_transfers(external_host,
                                          files=job['files'],
                                          transfertool='fts3',
                                          job_params=job['job_params'],
                                          timeout=timeout,
                                          user_transfer_job=user_transfer_job)
```

The workflow inside **Rucio** will be as shown below:



3. Testing / Commissioning:

I will have to write some unit test based on the **unittest** module in **Python** as required during the implementation and integrate them with the **Rucio-dev** tests.

Testing of Task 1:

In this task I will have to test the various methods and helper functions that were implemented in this task by calling out them locally and check their working on the provided endpoints. This testing has to be done properly as this is base of the entire transfertool inside Rucio.

Testing of Task 2:

Here I will have to test the validity of the provided endpoints while the transfer action is called inside **rucio.core.transfer** and also validate the client credentials inside **rucio.cfg**.
After integration of the **GOTransferTool** interface inside **rucio.core.transfer** and **rucio.core.request** I will have to test these methods locally to check whether the transfer functionalities are working as expected and to find out any loopholes in this integration.

Testing of Task 3:

After integration with the **conveyor-submitter** my next testing task will be to conduct a transfer using the **Rucio API** or **CLI** by creating a **GO RSE** and then to ask mentors to provide endpoints where we can conduct transfers in experimental conditions**.**

5. Timeline

Duration	Task
April 9	**Deadline for Student proposals.**
April 9 - May 6	Reading the **Rucio** documentation and try to experiment more with it.
May 6 - May 27	**Official Community Bonding Period:** Getting more involved into the **Rucio** community and trying to get familiar with the **Rucio** codebase. Knowing about various projects going at **CERN** and projects where **Rucio** is involved. **Begin with implementation of Task 1:** ● Start coding the basic methods as mentioned in the design. ● Will have to ask mentors for access to test **GlobusOnline** endpoints owned by **ATLAS** to complete the tests successfully. ● Test these methods by calling them out locally and take input as given during actual execution.
May 27 - June 10	**Official Coding Period Starts:** ● Evaluate and discuss with mentors which additional methods will be required and also find out the required helper methods. ● Complete coding the additional and helper methods. ● Setup the authentication against **GO** as mentioned in the implementation flow. ● Re-perform the tests again by the testing the additional methods and the helper methods on the provided endpoints.
June 10 - June 24	● Take inputs from mentors regarding the code of the transfertool and make changes as suggested. ● Run the local **Rucio-dev** tests.

	● Additional time to solve any unexpected problems and buffer for any unexpected delays. **Finish with Task 1.** This task will add a new interface named **GlobusOnline** to the transfertool kit and complete the design and implementation.
June 24 - June 28	**Official Phase 1 Evaluations.** ● Send a PR to the **Rucio** repository on GitHub along with the **Task 1** code.
June 28 - July 12	**Begin with implementation of Task 2:** ● Start using the **GlobusOnline** interface inside the **rucio.core.transfer** and **rucio.core.request**. ● Add **globus** functionality as it is for the **fts** one inside the methods in **rucio.core.transfer** and **rucio.core.request**.
July 12 - July 22	● Test the written code by calling the methods inside the files locally, evaluate the code with the help of mentors for issues in the code and fix them. ● Run the local **Rucio-Dev** tests. ● Additional time to solve any unexpected problems and buffer for any unexpected delays. **Finish with Task 2.** This task will integrate the **GlobusOnline** transfertool interface with **rucio.core.transfer** and **rucio.core.request**.
July 22 - July 26	**Official Phase 2 Evaluations.** ● Send a PR to the **Rucio** repository on GitHub along with the **Task 2** code.
July 26 - August 9	**Begin with implementation of Task 3:** ● To start the integration with the **conveyor-submitter**. ● Test the methods to get the

	rse-attributes locally so that actual implementation gets simplified. ● Add the methods from **rucio.core.transfer** inside **conveyor-common**. ● Then add the splitting point for the call of the correct transfer tool inside the conveyor-submitter. ● Test the working of the code locally, further evaluate the code along with mentors to find and fix any potential issues. ● Run the local **Rucio-Dev** tests. ● Start testing the transfertool inside actual **ATLAS** environment which supports **GlobusOnline** as transfer option. ● Try to solve issue **#121** depending on the amount of time left.
August 9 - August 19	● Find out any issues from the performed tests in the actual environment and solve them. ● Ask other users to use the **GlobusOnline** based transfertool. ● Additional time to solve any unexpected problems and buffer for any unexpected delays. **Finish Task 3.** This will fully integrate the **GlobusOnline** transfertool with **Rucio** and will support **Rucio** internally.
August 19 - August 26	**Official Final Evaluations.** ● Send a PR to the **Rucio** repository on GitHub along with the **Task 3** code.

6. Deliverables

- A working **transfertool** based on **GlobusOnline** which will be fully functional and useful for **Rucio** based transfers.
- The **GlobusOnline** transfertool will provide functionality equivalent to the existing **FTS3** based transfertool.

Future Goals

- To update the transfertool code with updates from the **GlobusOnline** SDK.
- To work on this issue which is more deeper than my project and also simplifies the **Rucio** transfertool workflow.
- To continue working with **Rucio** community and help in building **Rucio** further more.