Task 4.2. Ideas/Activities

Objectives
O4.1 Enable integration of computing resources from user allocations into the EuroScienceGateway
O4.2 Enable integration of existing user storage into the EuroScienceGateway
O4.3 Deliver a distributed caching mechanism across the Pulsar providers
O4.4 Implement a smart job-scheduling system to optimise the job distribution across Pulsar and user allocations.

WP4 Description:
The current Pulsar Network proved highly effective during the Pandemic. Hundreds of thousands of SARS-CoV-2 datasets could be distributed across Europe and compute resources could be scaled as needed. However, the configuration is currently not scalable as inclusion of storage and compute resources requires action by the system administrator. This work package will develop the needed extensions of the EuroScienceGateway components (Pulsar and Galaxy) to automate and facilitate the integration of user provided computing and storage resources. As the Pulsar Network grows with new providers, a smart job scheduler will ensure an optimised job distribution across all available computing systems for each user.

Task 4.1 Bring Your Own Compute (BYOC) (M1-24) (Aligned with objective O4.1)
This Task will develop an integration point, where a user can add compute resources to the personal Galaxy user account, e.g. AWS resources, or a Pulsar instance as an interface for local computing resources (M4.1). In both cases the user adds those additional endpoints to the user profile and establishes a secure hand-shake between Galaxy and personal Pulsar resources. Those resources are then taken into consideration by the smart meta-scheduler developed in T4.3. Any user of the EuroScienceGateways will be able to “bring their own compute” (BYOC) resources if the user has some cloud credits, or HPC access credentials, but no running Pulsar server nor the expertise to set one up. The task will leverage the reproducible and automatic deployment of Pulsar endpoints with optional computer clusters (developed in T3.1) to trigger the deployments from the Galaxy user-interface using libraries like cloud-bridge.

Task 4.2 Bring Your Own Storage (BYOS) (M1-24) (Aligned with objectives O4.2, O4.3)
Similar to T4.1, this Task will enable the addition of own storage resources to the system (M4.1). This could be personal S3 buckets, OneData repositories offered by EGI, or data repositories using iRODS as interface. Users will be able to “bring their own storage” (BYOS) and make these resources accessible by the job submission system. This will not only provide ways for a user to extend their local quota limits, but also enable new access models, in which EOSC provides storage solutions that can be seamlessly integrated into the EuropeScienceGateway. This task will also implement *a mechanism for data locality based on a caching layer that will track which (Pulsar) site a specific dataset is already available or which site is geographical near the data so the job scheduling decision process (T4.3) can choose compute resources with “nearby” storage.*

Task 4.3 Implement a smart job-scheduling system across Europe (M7-30) (Aligned with objectives
O4.4)
A job that enters the EuroScienceGateway needs to be routed to a given compute resource available for its execution. This task will extend the DIRAC meta-scheduler for smart distribution of computing jobs in the available resources, either from the existing Pulsar Network or from those user-provided allocations enabled by T4.1. The task will implement new features into DIRAC to support Pulsar as a jobs execution framework and to optimise the location of each task considering data locality (T4.2) and user preferences (e.g. carbon-footprint, performance, cost) (M4.2). The meta-scheduler will use a model for optimised job scheduling that will be trained using job profiles (e.g. memory and CPU requirements) and resources profiles that will be provided by the different Pulsar endpoints in the network (CPU and disk-IO performance, available accelerators like GPU and FPGA). Profiling will be automated by collecting metrics from compute resources as the jobs are executed in the available infrastructure. The meta-scheduler model will provide recommendations for different optimization strategies, like cost, job runtime or energy efficiency. Finally, the meta-scheduler will increase the robustness of the EuroScienceGateway by rescheduling jobs in case a Pulsar endpoint is temporarily unavailable or a job is crashing on insufficient resources.

Preliminary work:

- BYOS is effectively being implemented by some of the Galaxy core developers:
  - https://github.com/galaxyproject/galaxy/pull/15875
  - https://github.com/galaxyproject/galaxy/pull/14073

Ideas:

1. **Integrate OnedataFS as a files source plugin:**

   First, we need to prepare w Python wheel package for it (we are looking into that), and then try to integrate it into the Galaxy codebase. With the latter, I think it would be nice to meet for some pair programming sessions.

   Questions:
   How do deferred datasets work (file source plugin) from the technical point of view? When a user chooses a dataset, some kind of reference to it is stored? What is the format of this reference? How is this later resolved and is it done via the file source plugin or object store?

2. **Integrate Onedata as a new object store**
   There are 4 possible approaches:
   a) Integration with Onedata's REST API
   b) Integration with Onedata's S3 interface - seems to be the fastest way to build a PoC. We can subclass the generic S3 object store and add additional Onedata-specific capabilities.
   c) Using OnedataFS (pyfilesystem plugin) - here we also depend on the ability to package it as a wheel. **Or can we use a conda package here (as an object store)?**

d) Using Oneclient (POSIX mount based on FUSE) - **PROBABLY** would work for one object store, not for BYOS (it has to be instantiated for each user as an external app, would become a dependency of galaxy).

Questions:
What are the most commonly invoked operations of an object store and how big are typically the files transmitted? It would be good to know this to choose the best integration approach. How should we write the integration tests? They seem to be quite selective.

3. **Enrich object stores with locality information, to be used during scheduling**
    - Pulsar runners should be aware of the locality info
    - Onedata can provide locality information per file
    - Object store interface should be extended with an abstract method to resolve data locality information about a dataset
    - Other object stores could provide static locality information - one idea is to tag object store definitions with locality information
        - Static locality information in the object store definition → An external config file which maps object-store IDs to GeoIP
        - Users would have to tag their own object stores (BYOS)
            - Admin can enforce this through template
        - Locality tags not necessarily aligning with a set of allowed locality values. There could be tags from user perspective and from the scheduler perspective.
    - To maximize the use of data locality, all object stores could be virtualized under a single Onedata object store, exposed to Galaxy. This could be done for the ESG/.EU deployment only, just a specific case of the general approach, where an object store implements static or dynamic data locality information.
    - Questions:
        - What is the format of locality data? Geolocation? Site name? (having a closed set of storage sites?)
            - What information does the scheduler want to extract from the data locality? What is relevant for making decisions?

Actions:
- Schedule meetings:
    - Bjorn and potentially other devs to discuss the ideas of implementing the OneData backend in Galaxy and tagging Galaxy backends with locality information
    - With people involved in T4.3 to discuss how the meta scheduler can make use of locality information either through the tags or from OneData
- Propose ideas to Galaxy core developers potentially during the weekly meetings (every Tuesday at 4pm)
- Ask feedback from Galaxy devs for using Onedata's S3 interface to implement Onedata as a Galaxy backend
- Involve Bjorn for feedback on ideas and EGI conference
- Create Galaxy PRs

Minutes

- Previous discussion notes

22-05-2023

- Scheduler probably doesn't have a concept of geolocation for example (or it might have, and what then does that look like - to be found out) .
- Data locality info of a dataset must be resolved by Galaxy as a pre-execution step and added to job definition. It may use it in destination mapping to make a smart decision where to schedule the job. It is then passed to the job runner along with the job definition.
- DIRAC is a "virtual" destination that does not have a specific geolocation, since it's a layer that can have multiple compute resources underneath. When the DIRAC destination is selected, the job is redirected to the DIRAC job runner (configured for the destination). It uses the data locality info from the job definition to choose the best fitting site for computation.
- From the generic point of view, a job runner does not have to use the data locality info, and so doesn't the dynamic destination mapper.
- Will ARC try to use the data locality info?

Actions - we need to deliver a simple PoC:

- We start with simple geolocation (coordinates) as data locality info.
- Galaxy: Bjorn: lets create an external config file that maps the object-store IDs to a GeoIP and experiment with this. Similarly, we add geolocation data statically to destinations. TPV can then read this additional file and make some smart decisions.
- Onedata: implement an Object Store PoC that implements a function to resolve geolocation data for given dataset. Other object stores can use the static external config for now.