EOSC-hub

Report about the evaluation of OneData by the SeaDataNet community

Authors:	Jordan Maduro, Dick Schaap (MARIS) Gergely Sipos, Björn Backeberg, Andrea Manzi (EGI Foundation)
Date:	11/Mar/2020
Dissemination Level:	PUBLIC
Document Link:	https://documents.egi.eu/document/3583

Abstract

In this report we provide a summary of the findings of the OneData evaluation activity that was carried out by MARIS, member of the Marine Competence Centre of the EOSC-hub project during 2019. The evaluation was focussed on different data movement, access and caching use cases that are also described in the report.

EOSC-HUB RECEIVES FUNDING FROM THE EUROPEAN UNION'S HORIZON 2020 RESEARCH AND INNOVATION PROGRAMME UNDER GRANT AGREEMENT NO. 777536.

COPYRIGHT NOTICE

This work by Parties of the EOSC-hub Consortium is licensed under a Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/). The EOSC-hub project is co-funded by the European Union Horizon 2020 programme under grant number 777536.

Contents

1 Introduction

MARIS is technical coordinator of the SeaDataNet infrastructure and as such currently involved in the EU SeaDataCloud project. MARIS also represents SeaDataNet in the EOSC-hub Marine Competence Center (MCC). In this position MARIS regularly encounters software development challenges related to the distributed nature of the data and the large number of partners involved in the SeaDataNet network. A large portion of the challenges is related to the various aspects of data storage management. Currently, MARIS has multiple applications and systems created internally and by third parties, which make specific platform assumptions. For example, files locally available, synchronous file transfers, proprietary log formats, and log locations. These assumptions are difficult to overcome when these applications do not provide an interface to modify their behavior. A way to overcome these limitations is to develop specialized middleware or wrappers. However, developing these applications are time-consuming and often not reusable.

Onedata offers many advanced features related to data storage management. These features seem promising in solving many of our recurring challenges. By means of an amended EOSC-hub MCC workplan, MARIS aims to explore the opportunities available by leveraging the Onedata platform.

In this report, we examine the implementation of the use-cases proposed in the EOSC-hub MCC workplan.

1.1 Challenges

Each proposed use case illustrates an abstract challenge inspired by concrete cases we currently have or are faced with in multiple projects. The average number of files requested and processed is 500. Each file has a size of tens of KB but occasionally some larger files that average 500 MB are processed. A request is usually for ease of transfer and is usually between 50 to 100 MB. We list the Onedata concepts associated with each case.

2 Case 1: Cloud migration for legacy applications

To simplify the processes of migrating mostly desktop applications to the cloud we can use Onedata as a file storage abstraction layer providing seamless access to files distributed in the cloud environments. For this case, we used Octopus as the main application. We set up the testbed using Onedata version 19.02. We installed the Oneproviders using the onedatify installation script. The simple workflow consists of a number of steps.

Simple workflow

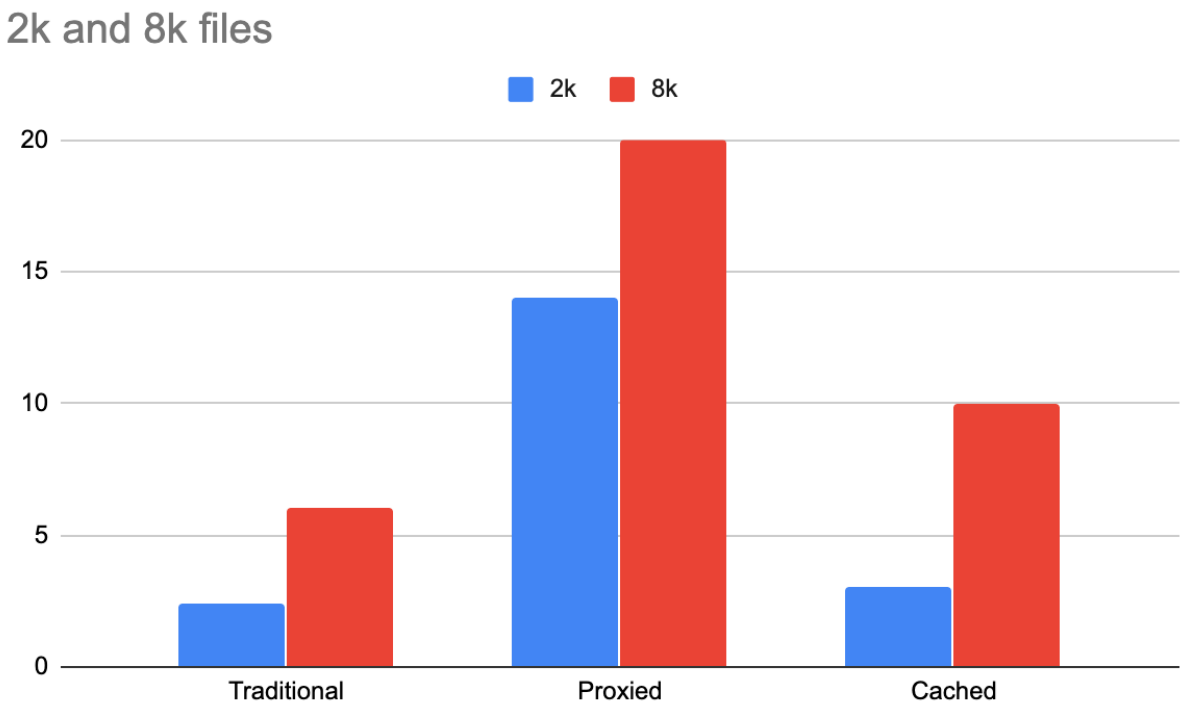
- Without Onedata
    - Download zip
    - Unpack zip
    - Run Octopus
    - Upload log
    - Run conversion
- With Onedata
    - Run Octopus
    - Move log
    - Run conversion

Setup

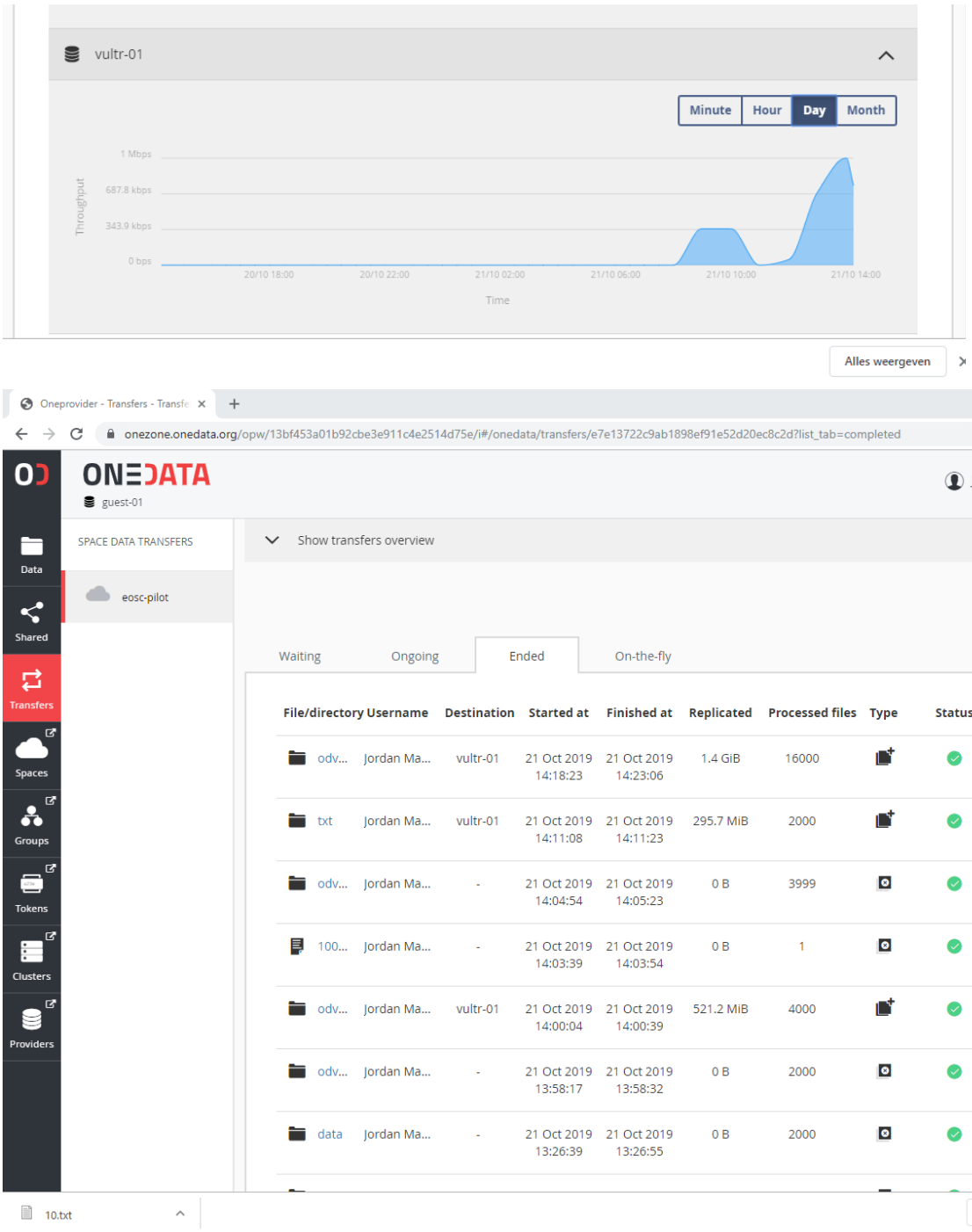
- **1 OneProvider with datasets**
- **1 OneProvider at computing site**

The Oneproviders are hosted in two different countries. The compute instance is hosted in the Netherlands and the storage instance is hosted in Poland. The experiment consists of multiple tests using different sample sizes of files. The workflow is semantically the same. A number of files undergo quality control by first using Octopus in batch mode to check the files. Then the files are converted to NetCDF.

In this case, we ran the workflow multiple times under different conditions. We benchmark the workflows using the time command. The benchmark compares the performance of proxied (uncached remote files), Oneprovider cached files and traditional. In the chart below, the average runtime can be seen in minutes.



We experimented with the replication API to fine-tune the optimized setup. By first transferring the files in bulk to the compute instance you reduce the transfer times significantly.



2.1 Discussion

Using Onedata for this use case greatly reduces complexity. Once setup onedata is very transparent and does not require any special modifications to existing programs. However, there are drawbacks. Using the unoptimized setup can be extremely slow depending on the available bandwidth. Every file requires a network call that takes considerably longer time than an I/O. This can be mitigated if we use the replication feature of Onedata. By replicating the dataset before running the workflow

you only add about 15 seconds overhead instead of the 10+ minutes using the proxied network calls. The replication of data before running the workflow saves a considerable amount of time. However, running through a Oneclient still has an overhead. In the case of ‘odv2k’ a little less than 1 minute.

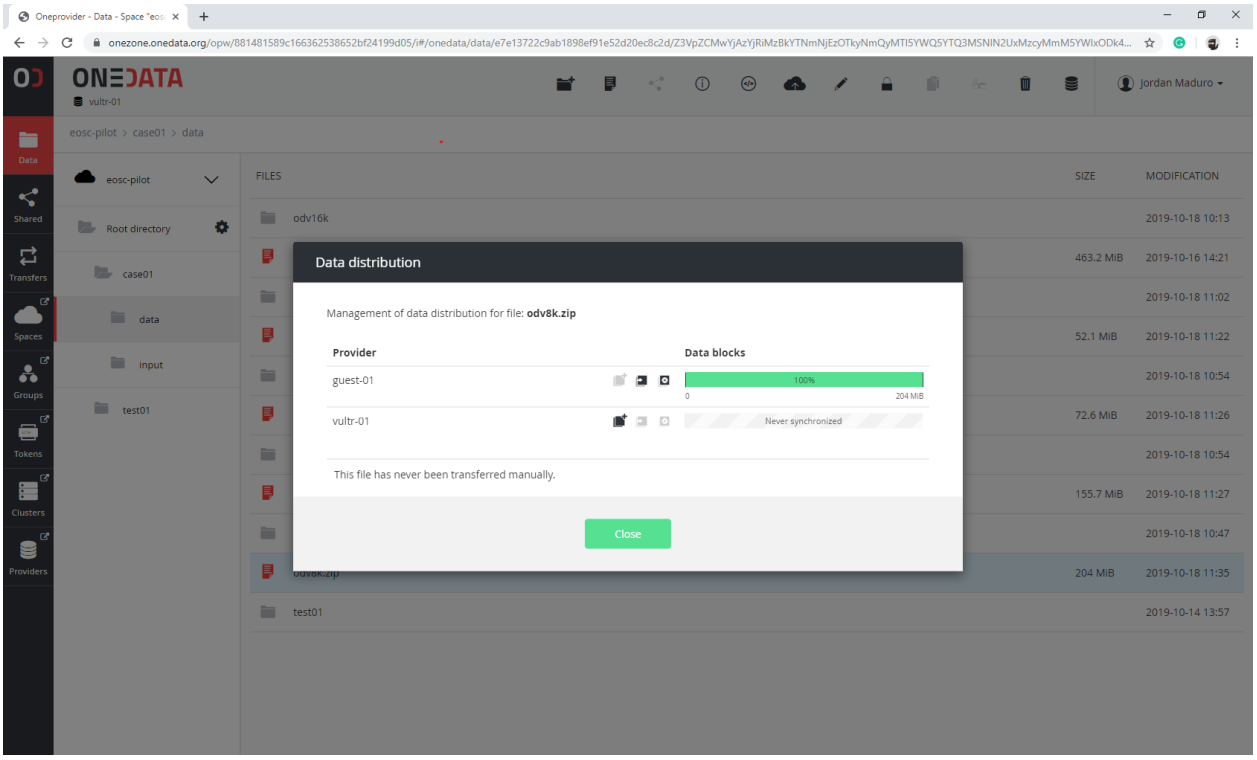


Figure 1 Data distribution before replication

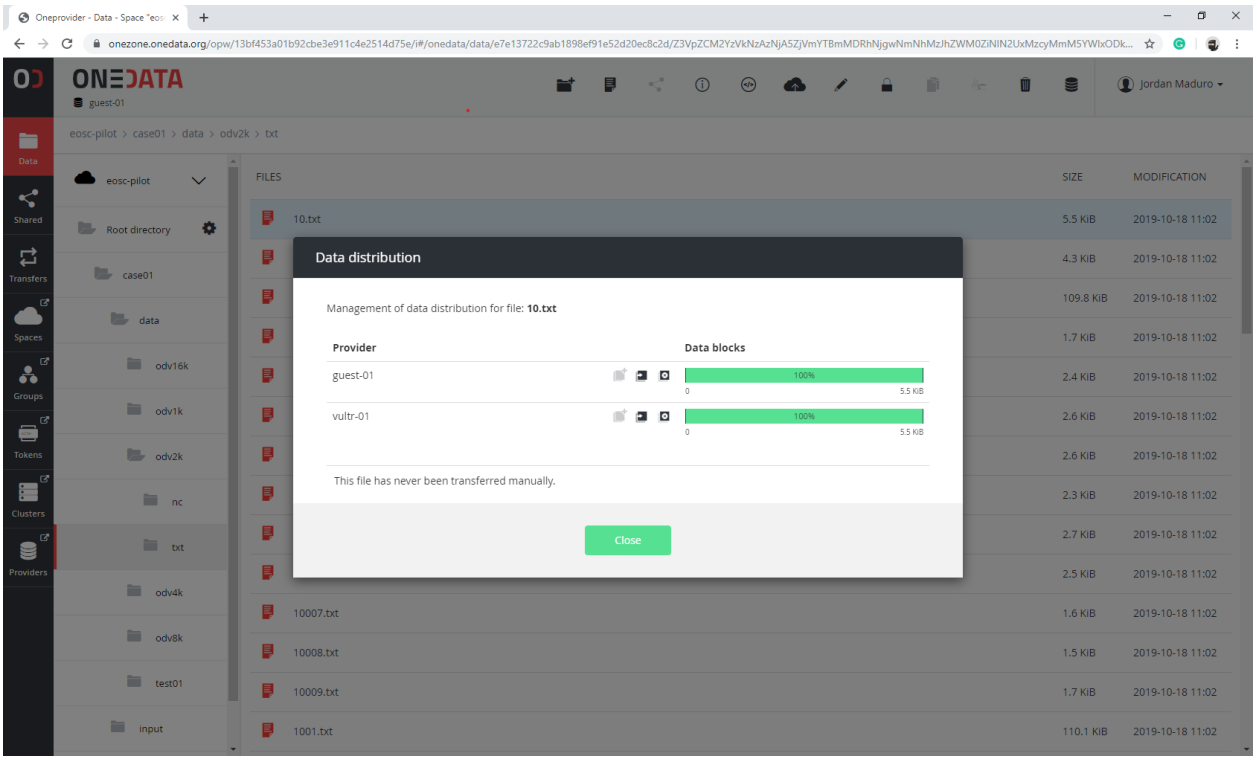


Figure 2 Data distribution after replication

2.2 Summary

Use case 1 is a great candidate for using Onedata because of the reduced complexity for the user. While Onedata is very sophisticated, it does not place any more cognitive load on the user. Thinking in terms of “removable storage” is enough to understand the basic concept of Onedata. We implemented the use case 1 by removing steps in the traditional workflow. The only considerations are if the overhead incurred by Onedata is acceptable and if the optimized setup can be used.

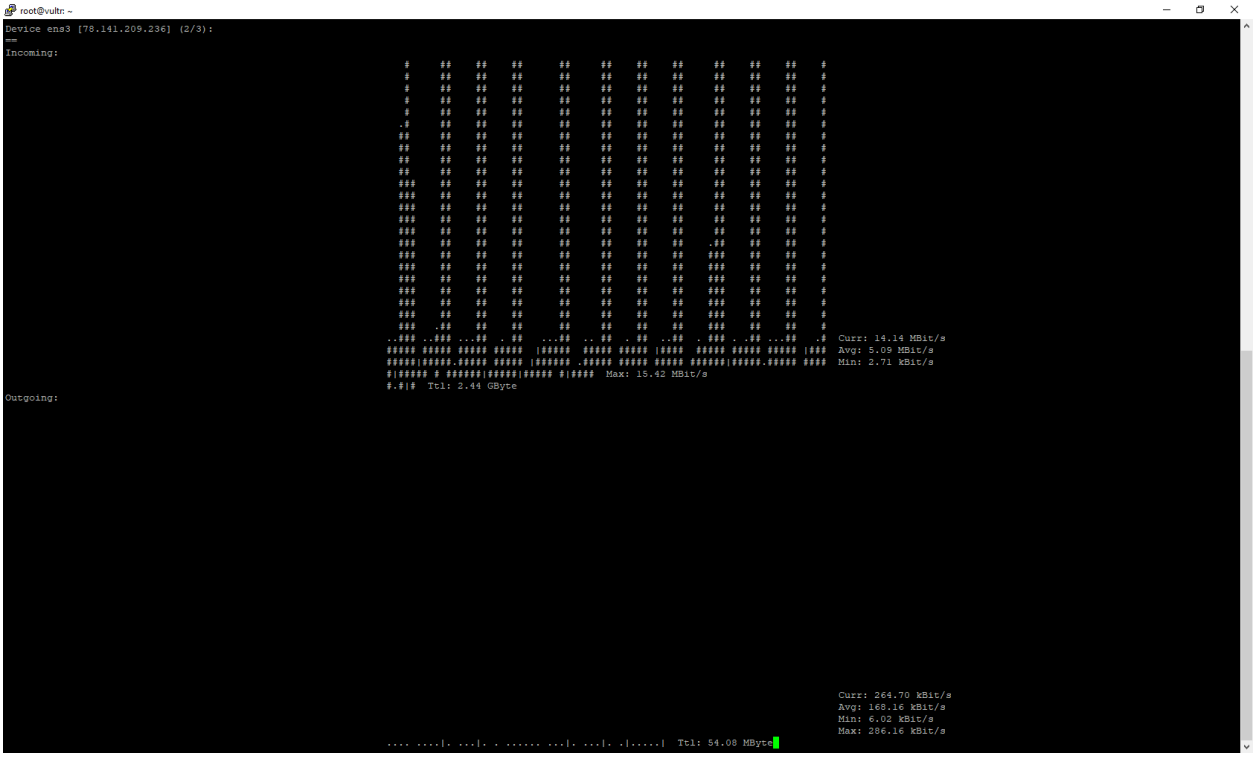


Figure 3 Network activity when running proxied workflow

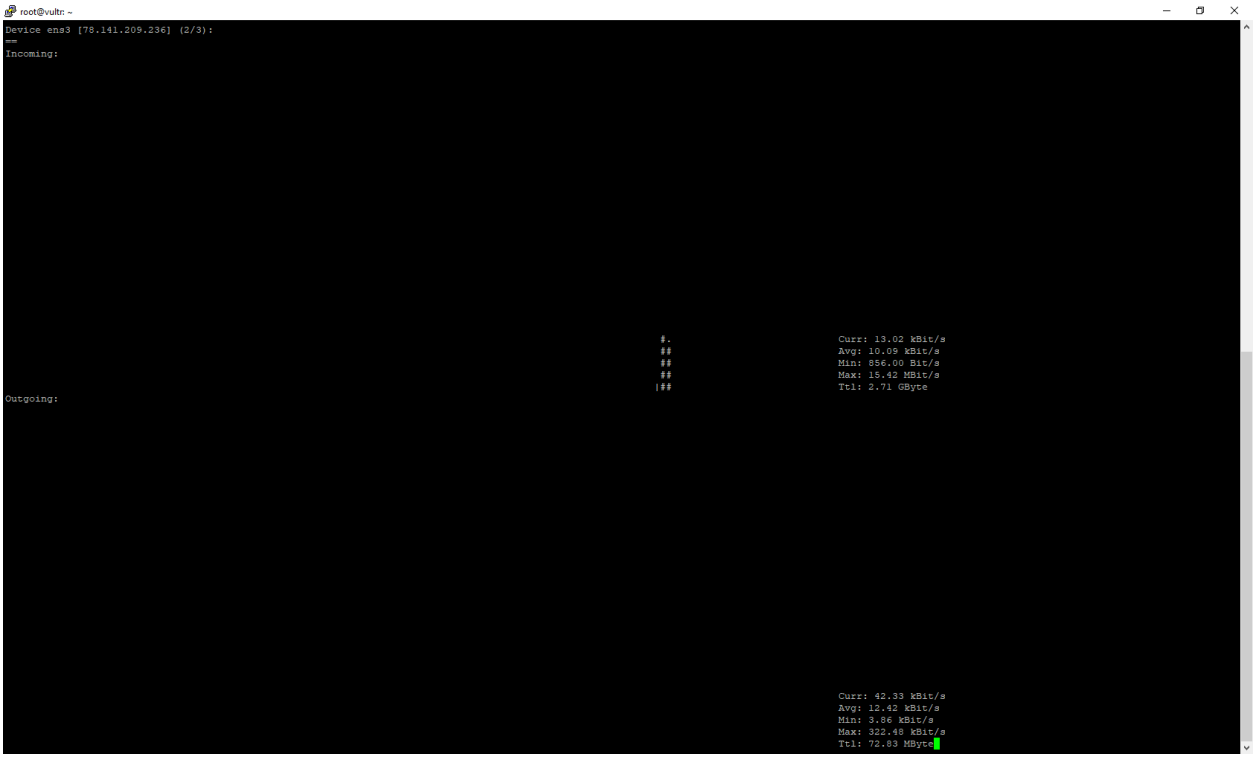


Figure 4 Network activity when running cached workflow

3 Case 2: Reducing redundant data transfer

Operations on large datasets are not always successful. In the case of quality control, certain datasets may be rejected and have to be revised before being submitted again.

This means that some complete datasets are transferred multiple times before being accepted. Using the direct access provided by Onedata to these files we can process only the required amount of data.

The implementation of this use case did not lead to a working solution. The functionality of viewing a zip file works. However, there is some unexpected behavior that made this use case hard to implement. To illustrate, the zip file is visible using the `ls` command. Which shows the files within the space, in this instance, located on a remote server. However, it is not found using the unzip -l command which lists the contents of the zip file. The command only works once you cache the file on the local Oneprovider. This defeats the point of using the Oneprovider to reduce redundant data transfer.

Another method that works is extracting the data on the remote Oneprovider. Then you can access each file individually. While it would not require files to be transferred multiple times it still transfers the uncompressed file. In the case of 500mb file zip, you get 3.4GB uncompressed files.

A possible solution is to first transfer the zip file to the compute site. After processing, any update operations should be performed on the remote site and OneProvider should sync the changed blocks. This approach was not attempted.

4 Case 3: Virtual space for user data storage and delivery

After a user searches for specific data he sends a request for a subset of datasets, a process is started to collect the datasets from many partners. This collection is an asynchronous process. The process can take weeks to collect all the requested files from the partners. This process is dependent on the resources available at the partners. We intend to use the features of Space and privileges management provided by Onedata to streamline these processes. We would provide the end-users access to his requested files through a shared space. Ideally, such a space can be used to make his requested files available for further processing in a cloud environment.

The Onedata platform offers the functionality to implement this use case. However, it was not possible to implement this use case due to a lack of resources, timing, and technical issues. The use cases were implemented using the latest version of Onedata. This version was only available at the Onedata.org Onezone. The partners were connected to the EGI Onezone which ran an older incompatible version of Onezone. This meant that the testbed created for the other use cases could not be used for this use case. Furthermore, technical issues arose during the migration of the EGI Onezone to the latest version of Onedata. The Onedata onedatify installation script was broken and therefore no new instances could be installed.

5 Case 4: Interface for distributed search using metadata queries

In order to find specific files in a distributed environment, we use proprietary search indexes. These indexes are inflexible and only allow querying of predetermined fields. This increases the time required to process and index all available datasets. With the current search interface, we process changes daily. However, the use of datasets within workflows would benefit from up to date information on the available datasets. To extend the discovery capabilities of a cloud application we can leverage the advanced metadata querying functionalities of the Onedata platform.

The implementation of the case started with the definition of views. The views are created using a MapReduce function to index specific fields in a space. In this case, we expect to have an array of subjects. Each subject is an array with 3 entries that represent the subject, object and units metadata fields in the ODV files. The function was written in a file and posted to the API using Postman. Once posted, the view would be available at the following URL.

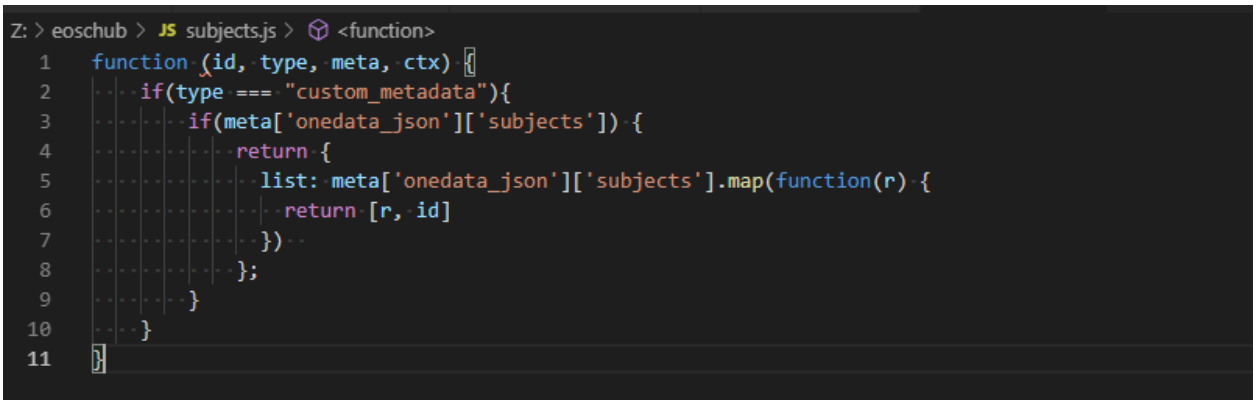


Figure 5 MapReduce function written in JavaScript

We then created a script to index files within a given folder within a space. This file is written in JavaScript and runs on Node v12. The script loads all files and then extracts the metadata lines, transforms and parses them to JSON. The JSON contains the 3 fields subject, object, and units. The JSON is then posted to the Oneprovider using the API.

After running the script we can query the API to see all the files that contain one or more combinations of the parameters. We first use Postman to query the view.

https://{{HOST}}/api/v3/oneprovider/spaces/e7e13722c9ab1898ef91e52d20ec8c2d/indexes/subjects/query?keys=[[

"SDN:LOCAL:AMON_7",

```
    "SDN:P01::AMONAAZX",
    "SDN:P06::UPOX"
  ],[
    "SDN:LOCAL:ASLV_52",
    "SDN:P01::ASLVZZ01",
    "SDN:P06::ULAA"
  ]]
```

Query URL

Which results in the following output. A list of JSON objects which contain a “key”, “value”, and “id” property. The most important property is the “value” property because this is the file id.

The file id can then be used to retrieve the file or the complete metadata.

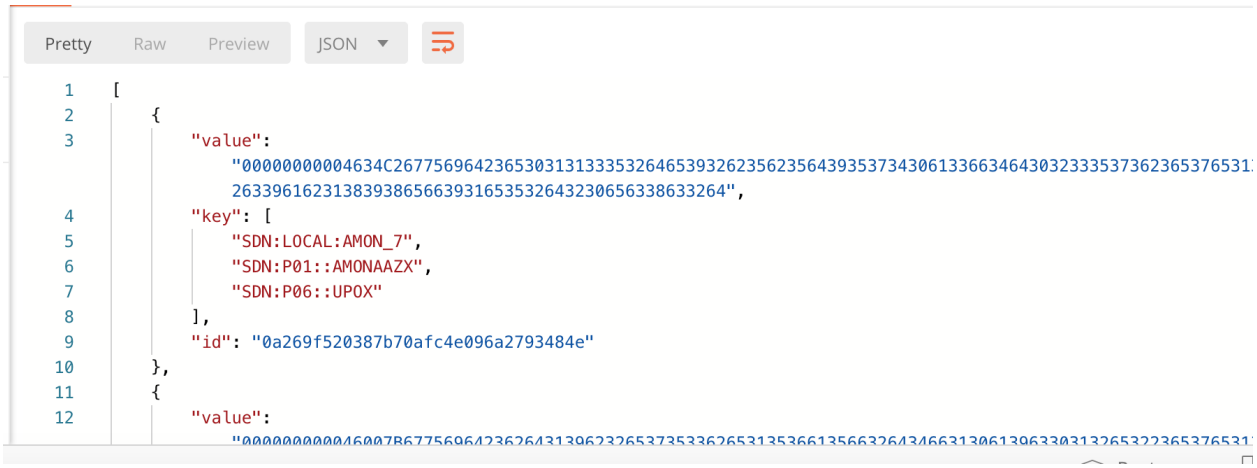


Figure 6 Pretty formatted search results

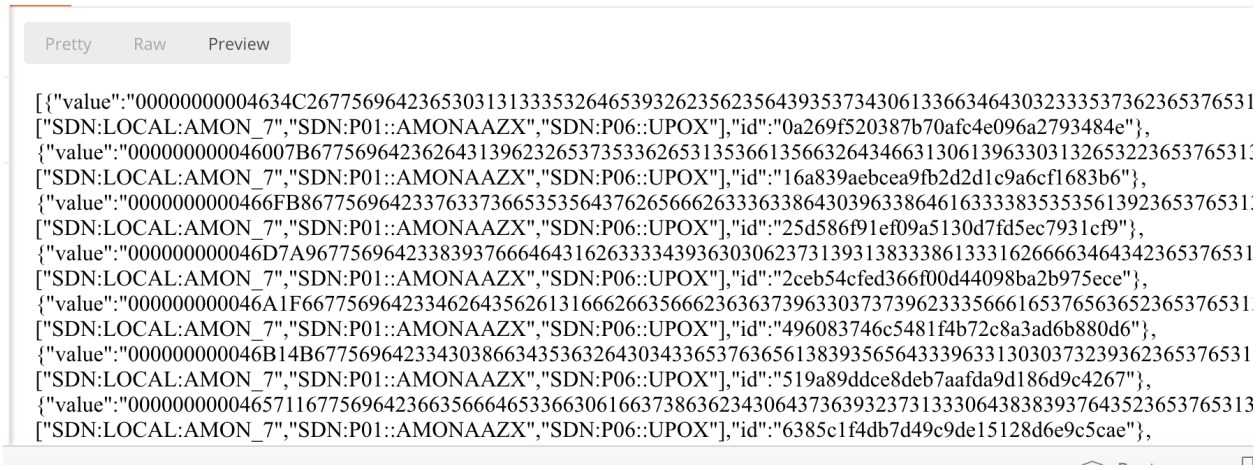


Figure 7 Raw search results

This use case depends largely on how the metadata is structured and added to each file. The primary method of annotation would be during the import of a file. Unlike the traditional method of having a central database with the metadata. The data providers can annotate their own files with metadata which can then be queried using the views.

6 Learnings/experience

The Onedata platform allows you to build workflows that eliminate explicit data transfer actions. Using it in the exact same way as the traditional workflows do not automatically provide a performance boost. It actually incurs a slight overhead. It does, however, reduce the complexity and the points of failure within a workflow. This helps in creating, maintaining, and understanding workflows.

Implicit cross space transfers can be very slow. Having your application request a file that is located somewhere else causes that file to first be transferred to your local Oneprovider. This implicit transfer is a great mechanism that reduces the complexity of a program. However, when your application requests thousands of files this can become a bottleneck. In this case, using the transfer API can reduce the amount of time spent transferring files by doing it in bulk.

Onedata is not geared towards handling files on a low level. It is intended to be an easy-access storage. The data should already be processed and organized in the desired way before uploading it to onedata. Otherwise, it can be a slow and tedious process. One way to bypass this limitation is by turning on the file system sync. With sync turned on and having direct access to the files' initial location will allow performing fs operations at native speed.

The Onedata platform is well documented and offers multiple tools to install and manage your Onedata ecosystem. Once installed the systems work seamlessly and have a lot of functionality out of the box. The only issue is the complexity of the installation. If the onedatify script is broken manual installation would be a challenge for even the most experienced users.

Optimized and Unoptimized setup

The unoptimized setup uses the Oneclient to connect to the space. This means that even though the files are available to the system they first have to be transferred before processing can begin. Subsequent reads suffer the same latency issue. This setup is only recommended for a very low number of files or prototypes.

The optimized setup is the same as the unoptimized version however the one difference is that there is Oneprovider installed on the system and connected to the space. The Oneprovider caches the files locally allowing subsequent reads to be at near-native speeds.

Vultr VM (Compute Instance)

CPU: 6 vCore

RAM: 16384 MB

Storage: 320 GB SSD

6.1 Unexplained behavior

The server suddenly showed an increase in CPU usage. This change happened without any traffic or operation and it was holding steady at about 80%. Using `top` we could see that the process `beam.smp` is the culprit. This behavior was monitored for two weeks and resulted in over 100% CPU average usage (162% to be precise). This irregularity was reported but not investigated further during the implementation of the cases.

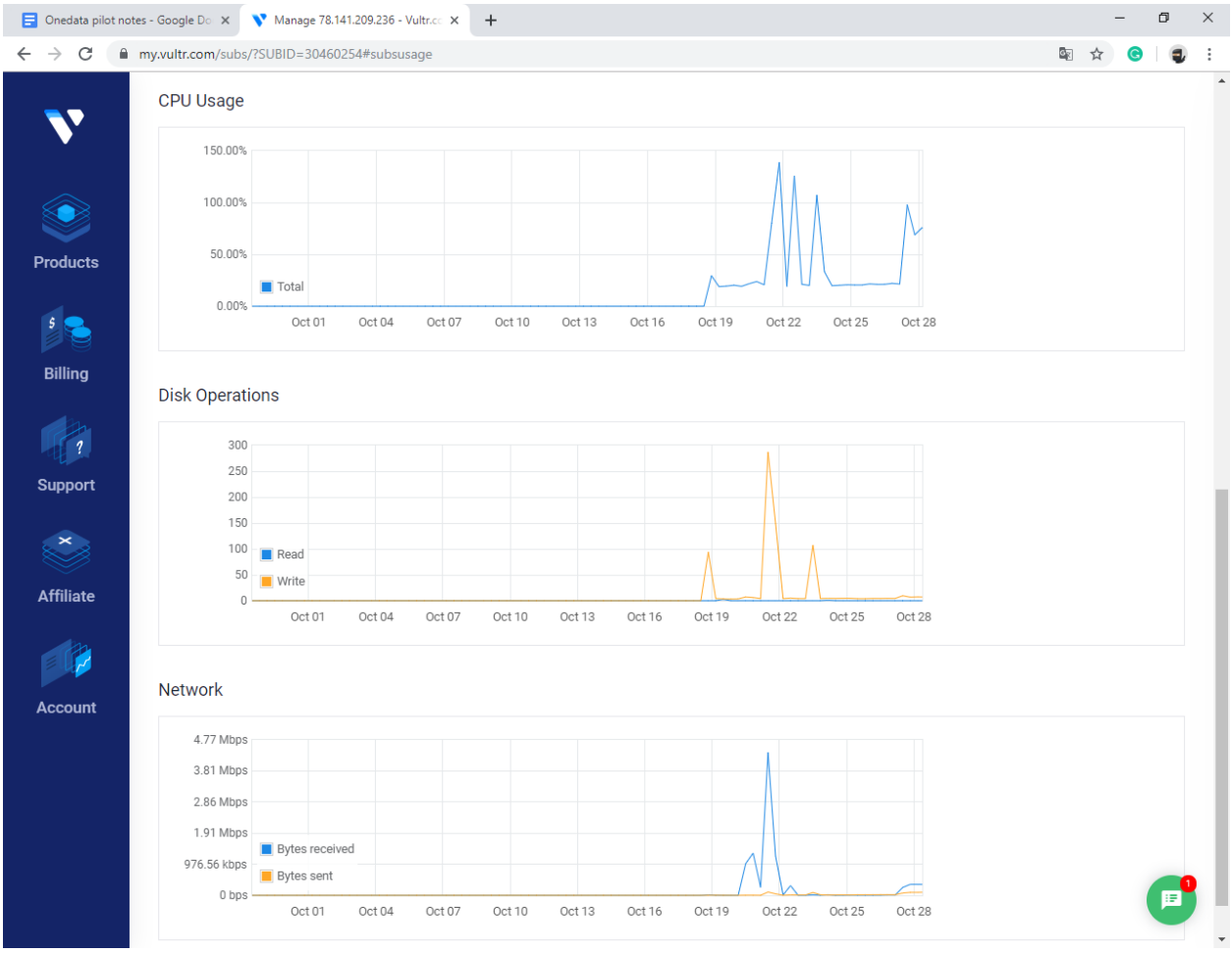


Figure 8 Unexplained CPU usage spikes

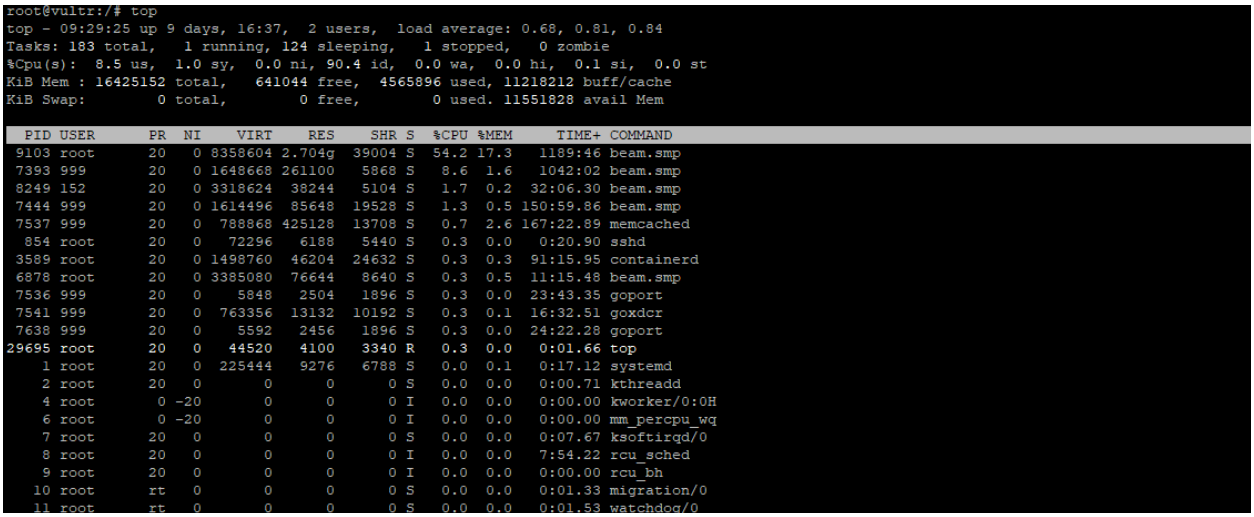


Figure 9 Unexplained CPU usage by beam.smp