Computing and Informatics, Vol. 39, 2020, 724–756, doi: 10.31577/cai_2020_4_724

PROCESS DATA INFRASTRUCTURE AND DATA SERVICES

Reginald Cushing, Onno Valkering

Institute of Informatics, University of Amsterdam
Amsterdam, Netherlands
e-mail: {r.s.cushing, o.a.b.valkering}@uva.nl

Adam Belloum

Institute of Informatics, University of Amsterdam
Amsterdam, Netherlands
&
Netherlands eScience Center
Science Park 140, 1098 XG Amsterdam, The Netherlands
e-mail: a.s.z.belloum@uva.nl

Souley Madougou

Netherlands eScience Center
Science Park 140, 1098 XG Amsterdam, The Netherlands
e-mail: s.madougou@esciencecenter.nl

Martin Bobak, Ondrej Habala, Viet Tran

Institute of Informatics, Slovak Academy of Sciences
Dúbravská cesta 9, 845 07 Bratislava, Slovakia
e-mail: {martin.bobak, ondrej.habala, viet.tran}@savba.sk

Jan Meizner, Piotr Nowakowski

UCC Cyfronet AGH
AGH University of Science and Technology Krakow, Poland
e-mail: {j.meizner, p.nowakowski}@cyfronet.pl

Mara Graziani, Henning Müller

University of Applied Sciences of Western Switzerland
HES-SO Valais, 3960 Sierre, Switzerland
&
Department of Computer Science, University of Geneva
1227 Carouge, Switzerland
e-mail: {mara.graziani, henning.mueller}@hevs.ch

Abstract. Due to energy limitation and high operational costs, it is likely that exascale computing will not be achieved by one or two datacentres but will require many more. A simple calculation, which aggregates the computation power of the 2017 Top500 supercomputers, can only reach 418 petaflops. Companies like Rescale, which claims 1.4 exaflops of peak computing power, describes its infrastructure as composed of 8 million servers spread across 30 datacentres. Any proposed solution to address exascale computing challenges has to take into consideration these facts and by design should aim to support the use of geographically distributed and likely independent datacentres. It should also consider, whenever possible, the co-allocation of the storage with the computation as it would take 3 years to transfer 1 exabyte on a dedicated 100 Gb Ethernet connection. This means we have to be smart about managing data more and more geographically dispersed and spread across different administrative domains. As the natural settings of the PROCESS project is to operate within the European Research Infrastructure and serve the European research communities facing exascale challenges, it is important that PROCESS architecture and solutions are well positioned within the European computing and data management landscape namely PRACE, EGI, and EUDAT. In this paper we propose a scalable and programmable data infrastructure that is easy to deploy and can be tuned to support various data-intensive scientific applications.

Keywords: Exascale data management, distributed file systems, microservice architecture

1 INTRODUCTION

We see application of HPC in both the scientific domain and industry: ranging from modeling global climate phenomenon to designing more efficient drugs. The state of the art in HPC is at the petascale (in the order of 1015 FLOPS), first achieved in 2008 [33]. However, we now see an enormous increase in the size of commercial and scientific datasets. Consequently, it is likely that the current petascale technologies will not be able to handle this and that we will require new solutions to prepare ourselves for the next milestone: exascale computing (1018 FLOPS). Exascale systems are expected to be realized by 2023 and will likely comprise of 100 000 interconnected servers; simply scaling up petascale solutions will likely not suffice [35]. Evidently, this raises many challenges in how the required hardware but also how to design applications that can make use of that many computing nodes. However, what is relevant for this paper is how the large volumes of data involved will be stored. Exascale systems will need novel Distributed File Systems (DFS), sufficiently scalable to accommodate for established DFS solutions.

The capacity of storage devices has been ever growing since the inception of the first hard disk drives (HDDs) more than 60 years ago. State-of-the-art technology currently limits HDD storage capacity for a single device at around 10 TB, with technology to support 100 TB HDDs expected by 2025 [36, 37]. However, many modern applications deal with data sets sized beyond what fits on a single machine or server. Moreover, these applications potentially require varying degrees of performance, availability and fault-tolerance. Applications are becoming more distributed in nature by pulling data from several different locations as in sensory data or by distributing computation to different locations as in edge computing applications. To facilitate this, we have to use a distributed data management system. The distributed data management system (DDMS) integrates into different file systems administered by different domains with the aim of creating a unified view of the user's data across administrative, geographic and technology borders.

In the PROCESS project, we proposed and developed a scalable and programmable data architecture that can be configured to best fit the data communication patterns of a given application and can be easily deployed. To achieve this goal, the PROCESS approach is to create a thin programmable layer on top of other established file systems with the aim to facilitate movement and pre-processing of data while minimizing state management so as to scale better. In order for the DDMS to actually manage data stored on multiple servers, these servers will have to communicate with each other over a network. Here, protocols used between any two servers could be customized and optimized for performance or any other attribute. From a design perspective, there are three properties that are desirable for a DDMS, namely: transparency, fault tolerance and scalability [39]. Transparency means that ideally the complexity of the distributed system should be abstracted away through the use of APIs. Fault tolerance means that in the event of a transient server failure (e.g. a failing HDD) or partial network failure (i.e. network partition) the system should continue to function, ideally without any compromising of data integrity. Lastly, scalability means that the system is able to withstand high load and allow for new resources (such as servers) to be integrated into the system with relative ease.

In this paper, we describe in detail the PROCESS Data Infrastructure and Data Services which aim to be a milestone in the path in the era of exascale computing. To put this work in context, in Section 2 we provide an overview of several established state-of-the-art solutions while also highlighting novel research projects and regarding them in an exascale computing context. In Section 3, we describe the architecture design processing starting from the requirement analysis, architectural decision and technology choices. In Section 4, we describe the implementation of the main important data services: LOBCDER, DataNet, and DISPEL, as well as all the mechanisms and APIs needed for their interactions. Finally, in Section 5 we describe the applicability to the different PROCESS use cases and derive a couple of scenarios.

2 RELATED WORK

There is an active research community with respect to improving DFS design, both in academia and in open-source communities. In this Section, we compare popular DFS (like the Hadoop file system (HDFS) GlusterFS, Ceph) in the light of three challenges we consider to be relevant for the development of the design of scalable DFS namely metadata management, and decentralization.

2.1 Scalability of Metadata Management

To ensure a future use of current DFS designs means that they not only have to be scalable in terms of actual storage capacity, but also in metadata management. The metadata scalability is of an extreme importance as half of the data processing operations are metadata operations [40], however, we have observed a lack of metadata scalability in most of the well-known DFS. Design of GFS and HDFS feature a single metadata server, Lustre allows for multiple metadata servers, but relies on explicitly storing the locations of files. GlusterFS somewhat improves in this aspect by not explicitly storing metadata regarding file locations but opting for algorithmic placement instead. It must be noted however that even with this in place, all the other metadata operations still happen on the data storage servers. The design of Ceph is probably the most scalable with respect to metadata management, since it allows for a cluster of metadata servers and also features algorithmic file placement and dynamic metadata workload distribution. A notable recent development in relational databases is NewSQL, a class of databases seeking to combine the scalability characteristics of NoSQL databases with the transactional characteristics of traditional relational databases. In a 2017 paper Niazi et al. present HopFS, a DFS built on top of HDFS, replacing the single metadata server with a cluster of NewSQL databases storing the metadata [41]. They attempt to address the issue of metadata management scalability by storing all HDFS metadata in a Network Database (NDB), a NewSQL engine for MySQL Cluster. They tested their solution on a Spotify workload (a Hadoop cluster of 1600+ servers storing 60 petabytes of data) for which they observed a throughput increase of 16–37× compared to regular HDFS. What makes this solution noteworthy is that it is a drop-in replacement for HDFS, allowing to be used in existing Hadoop environments, allowing them to scale beyond the limits imposed by the single metadata server approach. Using a similar approach, Takatsu et al. present PPFS (Post-Petascale File System), a DFS optimized for high file creation workloads. In their paper they argue that modern DFSs are not optimized for high file creation workloads, and that for exascale computing this can turn out to be a serious performance bottleneck [42]. They have evaluated their system against IndexFS (2014), a middleware for file systems such as HDFS and Lustre aiming to improve metadata performance [43]. With respect to file creation performance, they observed a 2.6× increase in performance. They achieved this by employing a distributed metadata cluster design using key-value metadata storage and non-blocking distributed transactions to simultaneously update multiple entries. Although only tested on relatively small clusters comprising of tens of servers, it is good to see that an effort is being made to improve upon aspects such as file creation performance, which might be a bottleneck in an exascale context.

2.2 Decentralization

In the solutions discussed so far we have seen various approaches to positively influence the scalability characteristics of DFS. A recurring concept is that of decentralization, distributing responsibility of certain aspects of the system to multiple non-authoritative servers instead of relying on a single or multiple dedicated centralized servers. Removing a single point of failure by distributing the workload should help to increase fault tolerance and scalability. We see such an approach in GlusterFS and Ceph, they both feature a decentralized approach towards the file placement. Here we will briefly discuss a recent project that seeks to go even further, a completely decentralized peer-to-peer DFS. Currently an open-source project with active development from a community of developers, the InterPlanetary File System (IPFS) is a DFS protocol designed to allow all connected peers to access the same set of files [44]. The author describes it as being similar to the Web in a single BitTorrent swarm exchanging objects within a Git repository. Removing all single points of failure by taking a completely distributed peer-to-peer approach is very interesting, since it in theory provides infinite scalability. However, having to rely on servers beyond your control likely rules it out for latency sensitive or mission critical applications. That being said, leveraging a globally distributed network of interconnected machines, such as DFS, is very relevant to at least capacity requirements. One can envision that given a large peer count, storing exabytes of data becomes almost trivial. Generally, we expect that the concept of decentralization will play a significant role in the development of future DFSs to cope with ever increasing scalability.

What are advantages to the design of GlusterFS? First of all, it offers POSIX file semantics, which means that it is mountable like any other traditional file system and adheres to strict consistency requirements. Secondly, its replication via erasure codes is a more space efficient way of replicating data than naively storing multiple copies. But the main advantage is the fact that the design does not feature a server explicitly storing file location metadata. With respect to scalability, not requiring a metadata server that can potentially be a performance bottleneck is a significant benefit. For certain workloads, a disadvantage of the design of GlusterFS is that it works on file granularity (as opposed to aggregated data blocks or chunks). Such a design can introduce more internal administrative overhead when for example replicating huge numbers of small files. However, we deem it likely that its approach of having a decentralized Namespace will manifest itself in exascale DFS solutions of the future. The design of Ceph, allowing for clusters of not only data, but also metadata and monitor servers provides it with excellent scalability characteristics. Currently, it is already being used by Yahoo to store petabytes of data and is chosen as the technology to prepare their infrastructure for storing exabytes of data [45]. This in combination with the level of customizability makes Ceph a good candidate for an exascale computing DFS.

There are several clear advantages to Lustre's design. The first of which is that it allows for clusters of data and metadata, like Ceph. Secondly, the handling of client requests and actual storage of data and metadata occurs on different machines. In terms of scalability this is a clear advantage since it allows for explicit control over how many servers to dedicate to the handling of client requests and actual storage. Similarly, availability can be customized by introducing redundant backup servers to a cluster. The number of files that are stored in a single object is customizable as well, which means that Lustre is not necessarily tied to a single type of workload with respect to file size. However, the lack of replication at the software level makes it a poor fit for failure sensitive commodity hardware, especially when the cluster size grows. That being said, its metadata and data cluster architecture, given hardware providing built-in redundancy and fault tolerance, make it a good candidate for an exascale computing DFS.

3 DESIGN OF THE PROCESS DATA INFRASTRUCTURE

3.1 Requirements

The development of the PROCESS data infrastructure and services is motivated by its use case applications from different scientific domains namely medical imaging, astronomy, Industrial (Airline domain), and Agricultural Observation and Prediction. All these applications are facing the data challenges either at this moment or will face data and compute challenges soon due to the expected increase of the data sets. A one size fit all design will not be able to fulfill all data requirements of the applications. Even if all use case applications required that the PROCESS infrastructure should be scalable, they have different requirements when it comes to the type of data to be managed by the infrastructure and the storage technology. In Table 1, we summarize the characteristics of the data sets used in 5 different use cases in the light of the NIST Big Data Interoperability Framework: Volume 1, Definitions [48], which reference to the Volume, Variety, Velocity and Variability as the main characteristics of Big Data. The data requirements for the five PROCESS applications are not exceptional; more extreme data management requirements are also reported for other exascale applications in the U.S. DOE reports published in

2016 like the one for High-Energy Physics (HEP), and Biology and Environmental Research. Both reports mention data access and movement as a key element in dealing with growth of datasets. For the HEP community, the Large Hadron Collider (LHC) at CERN will continue to be the largest producer, it is expected that in the future (roughly 2025–35) each HL-LHC experiment will transition from $O(100)$ petabytes to $O(1)$ exabyte of data. The Report states the infrastructure requirements for exascale of the HEP community for 2020 and 2025. The HEP community has developed data storage and movement services, ROCIO, to meet its needs in the 2020 timescale. In the Reports on the Biology and Environmental Research it is clearly stated that similar algorithmic barriers (lack of scalable solver algorithms and I/O) that challenged petascale performance will be faced again at exascale level, with the additional constraints introduced by accelerators and hierarchical memory.

	UC#1: Exascale learning on medical image data	UC#2: Square kilometre array/ LOFAR	UC#3: Supporting innovation based on global disaster risk data	UC#4: Ancillary pricing for airline revenue management	UC#5: Agricultural analysis based on Copernicus data
Volume	3.5 PB	$\sim$ 28 PB	1.5 TB (minimum)	$\sim$ 3 TB	10 PB
Variety	files [34]	files [49]	files	stream	files
Velocity [1]	low	low	low	medium	low
Variability [2]	low	low	low	low	low
Growth	2 TB/year [57]	5–7 PB/year	1 TB/year	1 TB/year	1 TB/year

Table 1. Main data characteristics of the use cases

The gathered common requirements are summarized in Table 2. The Data Services of the PROCESS projects implements them. Together with modularity and scalability, it makes its modules robust enough to support not only exascale communities coming from the PROCESS project but also supports a broad range of new exascale communities in the future due to the project's focus on reusability and sustainability.

Because the aim of PROCESS is to design a data infrastructure with exascale ultimate goal, we did study the exascale data storage landscape, one can see that a significant research effort is put into designing new hardware infrastructures at the

[1] Velocity is the rate of flow at which the data is created, stored, analysed, and visualized. Section 3.3.2, page 15, `https://bigdatawg.nist.gov/_uploadfiles/NIST.SP.1500-1.pdf`.

[2] Variability refers to any change in data over time, including the flow rate, the format, or the composition. Section 3.3.2, page 15, `https://bigdatawg.nist.gov/_uploadfiles/NIST.SP.1500-1.pdf`.

Requirement	Use Case	Service/Module
Integrated to access all the data storage centres	All	LOBCDER
Efficient and user-friendly data upload, transfer and download	All	LOBCDER (and its integration with IEE)
Provide access to storage resources on computing sites	All	LOBCDER
Provide support for HDFS	4	LOBCDER
Fast data transfer	2	LOBCDER (and its integration with Data Transfer Nodes)
Support pre-processing pipelines	1	DISPEL
Support various transfer protocols (e.g. GridFTP, SCP, etc.)	All	LOBCDER
Support meta-data management	All	DataNet
User-friendly interface for data access through the workflow management	All	LOBCDER (and its integration with IEE)

Table 2. Overview of the core requirements coming from the PROCESS use cases

datacenter level to optimize the HPC I/O stack [2, 3, 4, 5, 6, 7]. The DOE U.S. report [8] on the system requirements expected archival storage describes an emerging trend to embed more data management features directly into HPSS and thus acting as the storage level itself. Simulation tools are being developed to support the design of exascale systems and better understand the features and design constraints [9]. However, it is clear from many analytical studies [10, 11], which try to estimate current and future expenses in terms of energy consumption, predict that one single exascale datacenter is not realistic and thus the current effort of optimizing the HPC I/O stack has to be complemented with an effort to create a data management layer which can scale across data centers.

3.2 Design

3.2.1 Process Data System

To be able to claim that a data infrastructure is exascale enabled, it should be able to easily scale across geographically and institutionally distributed datacentres. This implies that the targeted data infrastructure is able to operate as a multi-system, on multiple data centers, multiple providers, multiple domains/types. Current approaches try to propose a data federation layer, which directly interacts with the backend storage, and for each backend they develop a specific driver in a plugin-like architectural style. They have been designed to operate in a specific setting that could be divided in 4 categories:

1. One system, one data centre, one provider, one domain/type like LHC;
2. One system, multiple data centres, multiple providers, one domain/type like WLGC, Astron, Globus;
3. One system, multiple data centres, multiple providers, one domain/type like cloud storage providers;
4. One system, multiple data centres, multiple providers, multiple domain/type like EUDAT.

On the contrary, the cloud approach has proven that scalability can only be achieved if we introduce a virtualization layer, which abstracts completely the details of the "hardware" infrastructure. New approaches based on Named Data Networking try to reduce the overhead in data transmission and will likely improve the communication within and across data centers [17, 18, 19, 20] and finding scattered across data centers could be completely agnostic of its location.

Following the cloud virtualization approach, we propose a data micro-infrastructure which is based on two basic widely accepted concepts IaaS and the fact that most secondary storages are accessed for read and write through a simple mount action regardless of the operating system or the storage type. As the variety of applications and collaborations between researchers increases, so do their dependences and requirements. Every group may have unique requirements and dependences for their applications. These different environments might require different data management, distribution and processing. Clearly, a one size fits all distributed system that tries to encompass all these different requirements beforehand will not perform well. Such an approach entails that the system needs to continuously resolve new dependences and requirements while also maintaining scalability. Furthermore, any smart data management is oftentimes very application or domain specific due to storage means (DB, files, etc.), different data access patterns, algorithm complexity, provenance, value, etc. This implies that the common data storage denominator between applications is, most often, raw block storage and a monolith system would need to handle all the different applications. A different approach that can handle the multitude of different data models, applications, distribution and management is through virtualization, by encompassing all these requirements in a data micro-infrastructure with specific nodes for handling the different aspects, e.g. a nextCloud node for sharing data within the group, and HDFS file system for computing, GridFTP for accessing remote files, etc. The whole infrastructure then becomes an ensemble of use-case micro-infrastructures each with its own full stack encapsulated in a virtual infrastructure.

Figure 1 illustrates the notion of a micro-infrastructure. Site providers provide raw resources through virtualization middleware such as OpenStack. They also provide raw storage that is accessible through the virtual machines. Through templating, micro-infrastructures can be booted up that will satisfy the groups' requirements for data processing. Cross provider data, process distribution and management are handled from within the micro-infrastructure. Cross group collaboration is also easily manageable, e.g., a group could give access to another group through their ownCloud node inside the micro-infrastructure. Scalability is improved since state management is divided between micro-infrastructures. One data management system will have difficulty managing exascale data, but many micro-infrastructures can better manage their own pool of data which is, most often, a few orders of magnitude less than an exabyte.

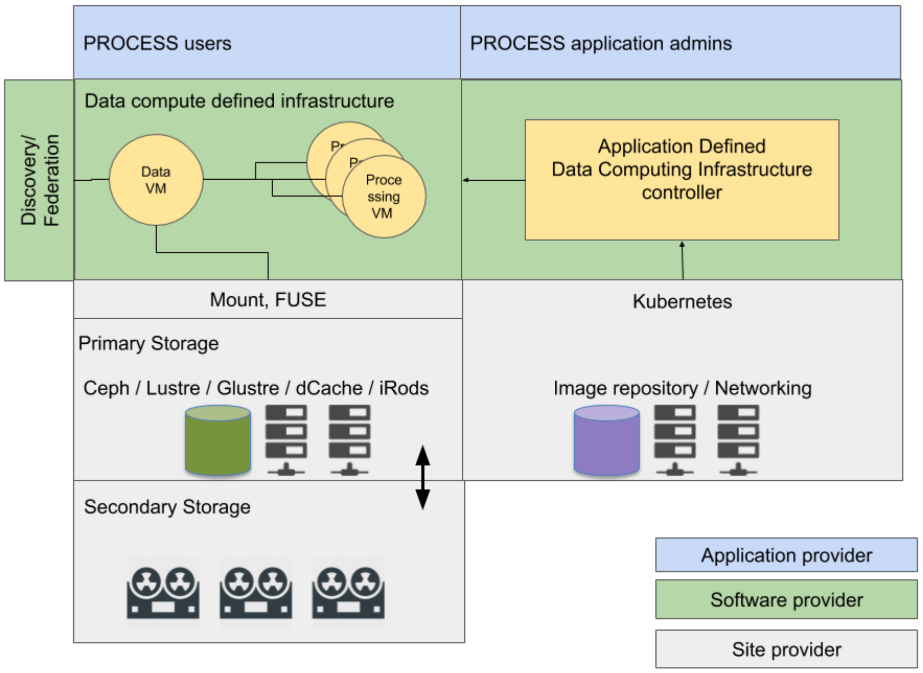

PROCESS users
PROCESS application admins
Discovery/ Federation
Data compute defined infrastructure
Data VM
Processing VM
Application Defined Data Computing Infrastructure controller
Mount, FUSE
Kubernetes
Primary Storage
Ceph / Lustre / Glustre / dCache / iRods
Image repository / Networking
Secondary Storage
Application provider
Software provider
Site provider

Figure 1. PROCESS micro-infrastructure

To better facilitate access to remotely stored large data sets, we have also included a component able to pre-process data remotely, at or near the place where they are stored, and to stream for further processing only a pre-processed, more compact data set. This component is based on the work of a previous FP7 research project ADMIRE [22]. It includes a decentralized network of services called Gateways, which are controlled by data manipulation programs described in a custom-designed high level language [23], and an expandable set of data manipulation primitives. A data process is instantiated as a network of streams of data through such manipulation primitives, which can load, filter, change, recalculate, clean, and store data (represented as a stream of uniform units of information, be it simple numbers, characters, or more complex structures).

4 IMPLEMENTATION

The PROCESS data infrastructure is composed of three main parts which are connected to data sources and managed by a service orchestration environment (see Figure 2). A core component of the environment is a distributed virtual system driven by LOBCDER. The tool has been rebuilt according to requirements coming from use cases several times ([50, 51, 52]). Its current version is based on a micro-infrastructure approach which allows creating a containerized micro-infrastructure of data services required by a use case.

It also has access to data sources via dedicated data adapters. The (pre)processing environment is driven by DISPEL which offers several processing elements (e.g. data access, data filtering, and data integration). DISPEL is accessible via DISPEL Gateway which is able to communicate with a WebDAV server via REST API. It accesses the data sources via dedicated data adapters. The whole data service environment is administered by LOBCDER which is connected with the service orchestration environment by REST API and WebDAV.

The PROCESS data infrastructure is meant to be programmable and customizable for every application. This implies that every application has its own set of data services that are deployed at runtime on the available storage resources. In this architecture LOBCDER takes the role of the manager which is responsible for instantiating the Data infrastructure for each application workflow. The other data services are instantiated as containers on-demand (depending on the application) to form Kubernetes pods. Service data containers instantiated by LOBCDER have different capabilities from access to remote storage such as HPC file systems to an interface which federates access to distributed storage (Figure 2).

4.1 LOBCDER/Micro-Infrastructure

LOBCDER implements the micro-infrastructure approach to develop the PROCESS data platform. The notion of a micro-infrastructure is to decompose large, monolith infrastructures into more scalable and manageable infrastructures. This decomposition allows for better scalability since state management such as indices, is split between many infrastructures. Furthermore, the increasing complexity of data requirements for applications necessitates a programmable approach that can be optimized for each application without interfering with other applications. For this reason, we leveraged the power of containers and created a platform using Kubernetes where users create an infrastructure with their own dedicated data services. Typical data services include data store adapters to connect to remote data such as HPC file systems, native cloud storage using Ceph block storage, runtime services that have access to the storage such as WebDAV points, Jupyter notebooks and data staging services.

The LOBCDER architecture is a hyper-converged infrastructure that provides a virtualized distributed programmable data layer. The infrastructure is a Kubernetes cluster using VMS and physical nodes distributed amongst PROCESS part-

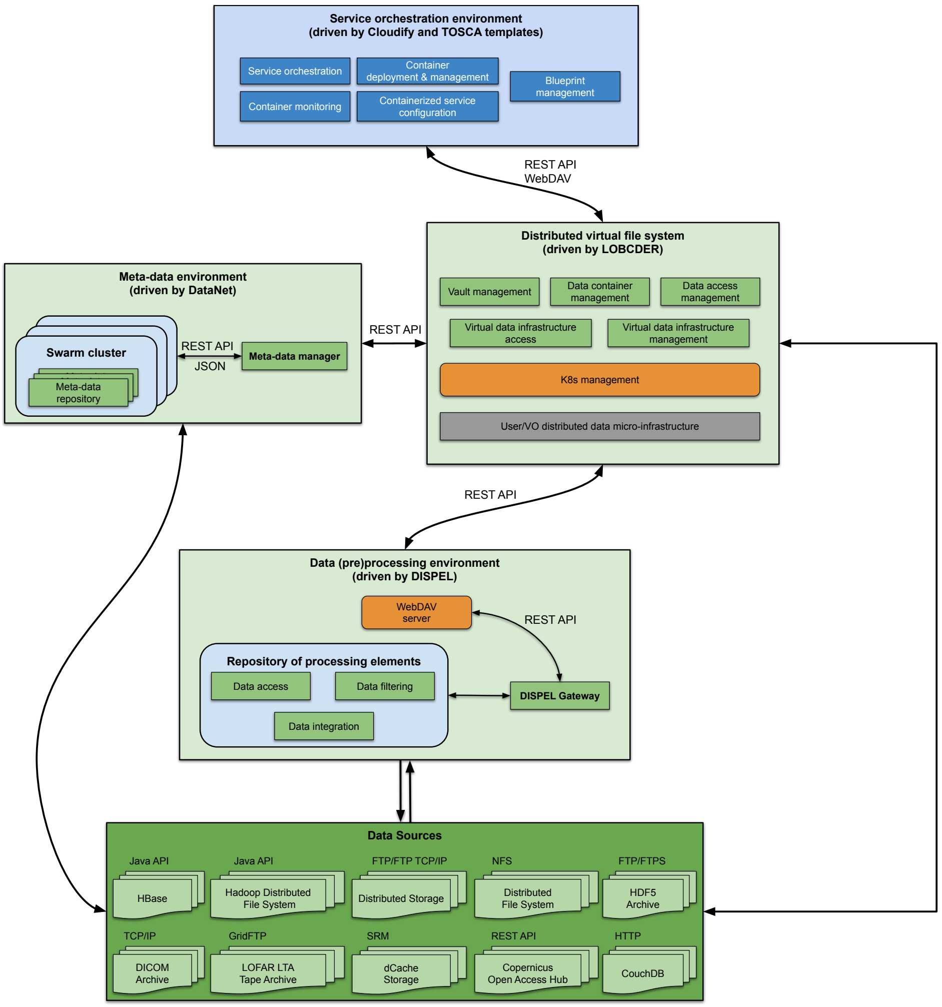


Figure 2. Process data service environment

ners. The VMs for Kubernetes cluster can be managed by the Cloudify orchestration service [21] that will dynamically instantiate the VMs in EOSC-Hub Federated Cloud infrastructure, configure and add them to the cluster. That will ensure the scalability of the micro-infrastructure and also may optimize data access, where the data service is located as close as the data storage the Kubernetes cluster servers a programmable layer to abstract data services and storage. Data sources can be of two types. The first type is the local-node storage, as is the case with dedicated data nodes. In this scenario a container has a persistent storage in the cluster which can be used as storage or cache for an application. The second type of storage is

HPC storage, in such case the data service containers mount remote storage in HPC clusters. The sequence to access and make use of the data services is described in Figure 3, the first two steps of the sequence shown in Figure 3 are dedicated to the creation of the micro-infrastructure:

Request a token: All the LOBCDER API calls are token protected. A token needs to be generated for users by the admin.

Create infrastructure: After getting a token a user needs to create his own data infrastructure through API calls with the header x-access-token set with the requested token.

Once the information about the created data infrastructure is ready, the execution environment can create and start the execution of the Application data processing pipeline.

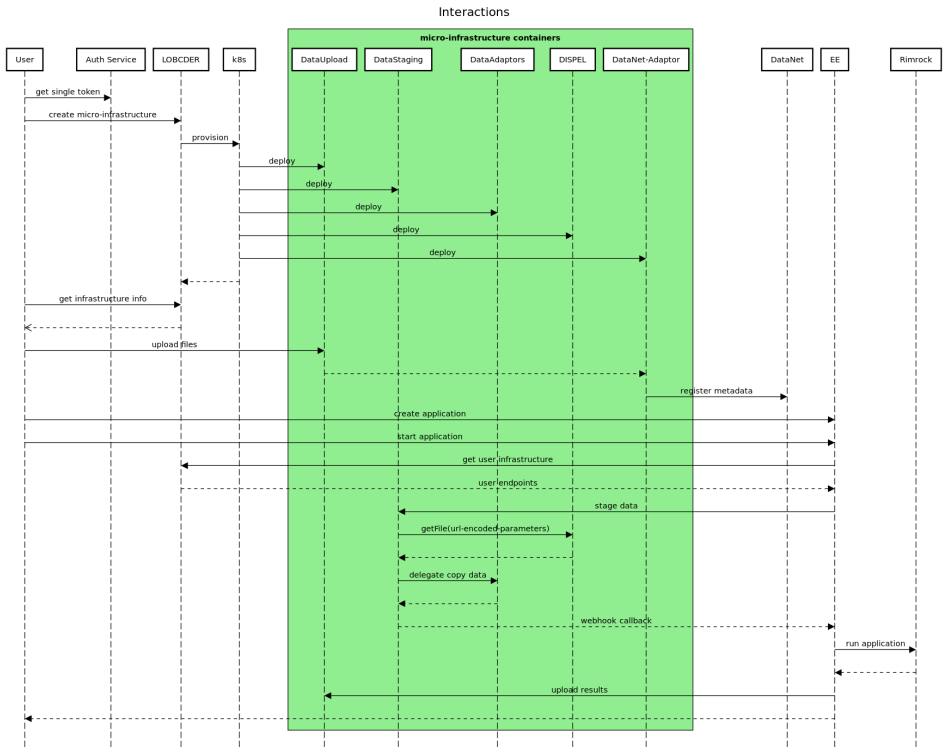


Figure 3. Sequence of interacting components from user perspective. The green block is a dynamically created virtual infrastructure per use-case. The infrastructure encapsulates use-cases' data management, credentials, distributed resources and pre-processing routines.

A REST API allows users to create their infrastructure as a set of pods and expose multiple WebDAV endpoints to access their data [53]. An important point to mention here is the integration with the Execution Environment (EE) which has to use the data services at several points during the application processing pipeline:

- The users' micro-infrastructure is dynamic thus services and their ports can change. For this reason, a first step of integration with EE is to discover the user's endpoints. This is done through the management API, specifically through the `/api/v1/infrastructure` call which returns a description of the endpoints.
- Every micro-infrastructure exposes an EE WebDAV specific endpoint which accepts tokens generated by the EE. Through this endpoint the EE environment can access all user's local and remote data through WebDAV.
- **Query and data staging service:** Every micro-infrastructure will implement a data query and staging service which will list the physical location of files and stage data onto HPC sites. This can be used by the EE to check the location of files on different HPC sites and also describe a staging pipeline with webhooks which will asynchronously stage in data (Figure 4) onto the HPC file system and use a webhook as a call-back to notify about staging progress.
- **Pre-processing workflows:** Often scientific applications have a pre-processing and staging workflow defined on the data services which will be exposed as endpoints whereby the EE can call and register a callback webhook to be notified when pre-processing and staging has finished so that computation can commence.

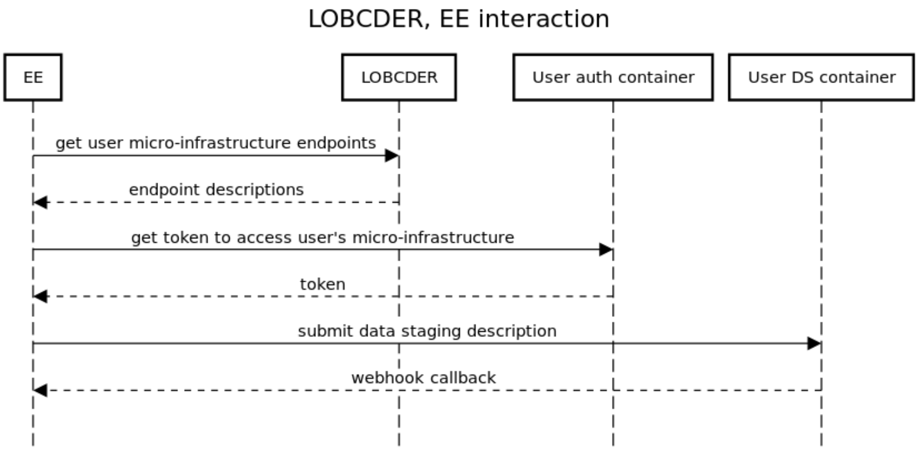

LOBCDER, EE interaction
EE
LOBCDER
User auth container
User DS container
get user micro-infrastructure endpoints
endpoint descriptions
get token to access user's micro-infrastructure
token
submit data staging description
webhook callback

Figure 4. Simplified EE to LOBCDER interaction. EE queries LOBCDER to retrieve user's dynamic infrastructure. EE can then access user's data services, e.g. data staging.

The LOBCDER micro-infrastructure approach revolves around containers. For this purpose, **several template containers** are developed for use in the PROCESS. Containers encapsulate different capabilities, from adapters to allow access to remote storage such as HPC file systems to user interfaces to access the storage. We categorize the containers into different groups depending on their capabilities. All the data containers identified from the requirement analysis fit two categories: the *logic containers* and the *Storage adapters containers* (see Table 3).

Storage adapter containers	Provide access to remote storage such as HPC file systems. **Examples**: sshfs, GridFTP, Cloud-native-storage, etc.
Logic containers	Provide a functionality on top of the storage adapters. **Examples**: token-based WebDAV, Jupyter service, and DISPEL service.

Table 3. The categories of containers to create any micro-infrastructure

4.2 Storage Adapter Containers

For this category we are considering three types of access to remote storage which are implemented as three storage adapter containers (see Table 4).

sshfs adapter container [24]	The container is able to mount a remote folder through ssh credentials. When creating an infrastructure through the API a user supplies his credentials to the remote server. The credentials are used to copy keys to the remote storage and are discarded after keys have been copied. This will allow passwordless authentication. A user can revoke access from LOBCDER at any time by removing the key entry in his home directory in `.ssh/authorized_keys`.
gridFTP container adapter	For high performance data transfers between sites we will employ gridFTP delegation service.
A cloud-native adapter	Whereby storage is provisioned directly in the Kubernetes cluster using Rook/Ceph storage manager. The storage is mounted into a container and exposed alongside the other adapters using WebDAV.

Table 4. Three storage adapter containers

4.3 Logic Containers

Logic containers help to develop/offer new services on top of three basic storage adapters (see Table 5). The logic container services are accessed by the user after the micro-infrastructure.

- User first needs a token to interact with the management API.
- User submits a JSON description of the infrastructure to the `/api/v1/infrastructure` url.
- LOBCDER will contact the k8s API to initialize a micro-infrastructure.
- The user queries the API to get the endpoint descriptions which include url and ports for the dynamically running services.
- The user can access the running services.

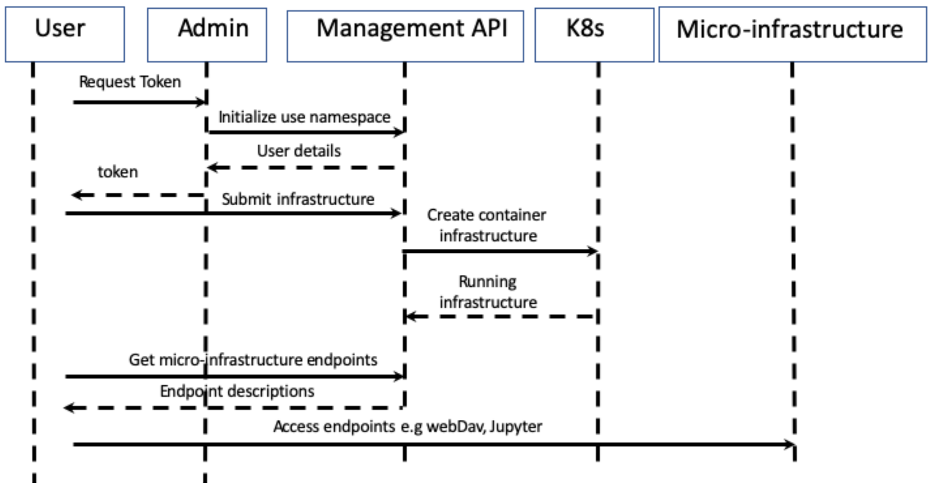


Figure 5. Simplified user to LOBCDER interaction. User requests credentials to access LOBCDER API. User can then submit requests to create virtual infrastructure and access services.

5 APPLICATION USE CASES

5.1 Process Data Service in Medical Use Case

The application setup of the medical imaging use case [28] has two computationally intensive workflows. The first, also referred to as Layer 1, consists of data staging and pre-processing. The second one loads the intermediate output generated by Layer 1 and focuses on the training of deep learning architectures, which needs

A WebDAV container [25]	Through the API infrastructure description, users supply a username and password for protecting the WebDAV point since this will be exposed publicly.
A token based WebDAV [25]	Meant for access by computing services. In this category we modified a standard WebDAV server to authenticate using web tokens. This mechanism of authentication is needed by the execution environment. When supplying the infrastructure description, a user also supplies the list of users with their public keys to be allowed through the WebDAV endpoint. This WebDAV implementation is expecting the WebDAV calls to have a header 'authorization' set with a token provided by an external entity (in/out case the execution environment). Upon access the WebDAV server will decode the header token check the user email is in the list of users and check the signature by decrypting using the public key provided when setting up the infrastructure.
Jupyter service container	Jupyter service container allows the user to access data through a processing environment whereby they can perform lightweight processing inside the data infrastructure. The adapter data is mounted in data folder on the container. In the following releases, this will be extended into a general user interface container for PROCESS with PROCESS-specific Python modules, and this general UI will be then extended into use case-specific UIs with more Python modules designed specifically to handle the use case data and pipelines.
Query/Staging service [26]	This service lists the files and their location on the adapter containers. The purpose of this service is to incorporate also staging capabilities for integration with IEE where IEE can request data staging between adapters so that applications would have just-in-time data on the HPC file systems (REF). These containers are meant to optimize application workflows execution, e.g. by scheduling data transfers between sites, caching containers with local-node data storage so frequently accessed data can remain easily accessible (to be developed as the project processes).

⋮

GPU compute nodes. This workflow requires fast access to the pre-processed data which means having the pre-processed data ready on the local file system before the compute can start. For this reason, we set up the pre-processing and staging workflow as part of the data services that will be able to pre-process and push the datasets directly onto the HPC file systems in preparation for the computation part of the workflow.

The pre-processing extracts patches from the high-dimensional medical images. A series of hyper-parameters are required as input to the runtime, such as the staging location of the data, the resolution level at which the patch should be extracted, the patch size and stride, and the patch sampling strategy (i.e. random sampling, importance sampling, dense coverage). The file system is scanned to retrieve patient-

⋮

DISPEL	DISPEL container gives access to remotely stored data and an entire data (pre)processing environment. The DISPEL data processing environment is currently available as a Debian-based virtual machine with a complete deployment of all tools, manuals and a tutorial with example data processes. The VM contains a graphical development environment based on Eclipse. Dispel is accessed via standard WebDAV interface (HTTP protocol) since it is part of the LOBCDER distributed data infrastructure. The parameters for data processing are encoded in the provided HTTP URL, as described previously in D5.1. The HTTP URL encodes the following parameters: (1) One selection of DISPEL data process description file template (in the DISPEL language). (2) Zero or more parameters to be filled in the template. Example of URL-encoded parameters: `http://lobcder.process-project.eu/dispel/tiffstore/312/20181129/12/0-1200-0-400`. Components of the URL are: • `http://lobcder.process-project.eu/`: URL of the LOBCDER WebDAV server • `dispel`: a prefix to recognize the sub-repository to contact (the DISPEL service) • `tiffstore`: selection of the DISPEL data process template to execute • `312`: subject designation (application-specific metadata) • `20181129`: data creation time (application-specific metadata) • `12`: layer in a multi-layer TIFF file (application-specific metadata) • `0-1200-0-400`: grid selection (application-specific metadata)
DataNet-adapter	DataNet-adapter container allows pushing of metadata to the Datanet service. DataNet allows performing operations on the metadata sets such as creating/updating/querying/deleting entities. DataNet is designed to offer straightforward user access via the REST API as well as GUI HAL browser. DataNet is available in the form of the Java source code under the OSI approved license as well as a Docker Container for the convenient deployment DataNet Rest API is described in Annex C
NextCloud [27]	Service for ease of use by users. Next Cloud container allows the user to view their data in dropbox fashion.

Table 5. Logic containers to support WebDav, Jupyter notebooks, and staging in/out data to HPC systems

related metadata and manual annotations. The physician annotations are used to build binary masks of normal and tumor tissue from the lowest image magnification level. From each of the two tissue types a set of image patches are extracted, by sampling locations in the high-dimensional image, according to the sampling strategy. Patches with non-relevant information (e.g. white content, black pixels, background, etc.) are filtered out and discarded. The pixel values of the image patches and metadata about the patient, the lymph node, the hospital that handled the acquisitions, the resolution level of the patch, the doctor annotations and the patch location in the image are stored in a HDF5 database.

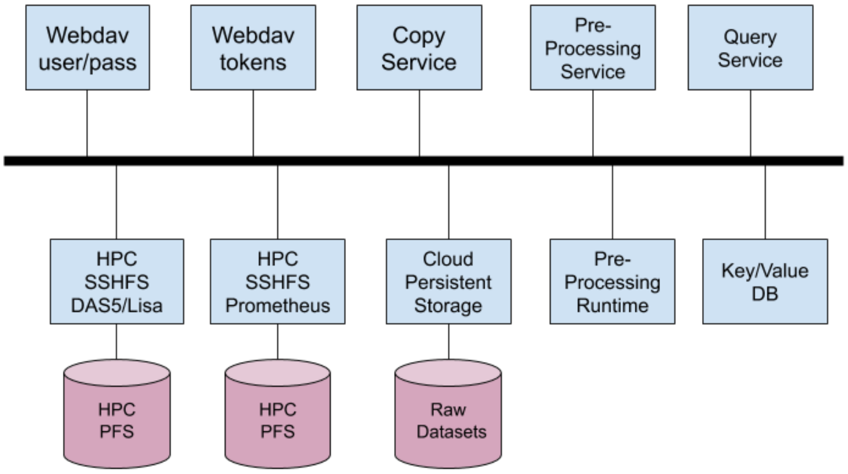


Figure 6. Micro-infrastructure for learning on medical image use case

In Figure 6, we illustrate the set of containers proposed for the data micro-infrastructure setup for UC#1.

WebDAV service: two WebDAV containers are used as a standard way to expose the data as a file system. A standard user/pass WebDAV can be used by standard WebDAV clients while the token-based WebDAV client is used by the Execution Environment.

Copy service: The role of this container is to expose a REST API that will handle copying files between sites or pull public files from the internet and directly onto the HPC file systems.

Query service: This REST service container will query all file systems to find where the physical file is being hosted. The query service is a precursor for the integration with DataNet.

Pre-processing service: This REST service container will allow users to define input raw data, input hyper-parameters to generate new pre-processed datasets and HPC output locations so that the pre-processed datasets are pushed directly onto the HPC file systems. It also keeps track of these generated datasets using a local database.

Pre-processing runtime: This container encapsulates the logic of pre-processing.

Cloud persistent storage: This container exposes a storage block hosted directly inside the Kubernetes cluster. This storage will be used to host raw data that is needed by the pre-processing pipeline and also act as a cache for the generated datasets.

HPC SSHFS: These are standard rudimental containers acting as adapters to the HPC file systems. Through these adapters, the copying service can push/pull data from the HPC clusters.

Key/Value DB: A container that maintains state such as indexes for the generated datasets and location of the files.

5.2 Data Services for LOFAR Use Case

Scaling is an important feature for the LOFAR Use Case [54] PROCESS data services have to be combined with this goal in mind. The pipelines should be executed by containers and when, say, two LOFAR archival observations are processed simultaneously, this can be enabled by doubling the number of containers – assuming these observations are of the same size. Simultaneous or quasi-simultaneous processing of multiple observations has the benefit of reducing latency induced by data transfers – i.e. staging of observational data, from tape to dCache and from dCache to a compute cluster – and by compute bottlenecks. Data transfers may take significant time due to the data sizes and distances involved. Even at 10 GBit/s, a 16 TB dataset will require about 4 hours to transfer. Fortunately, copying data from a temporary disk to the processing location may be done per observational subband. Thus, staging and copying can overlap. Compute bottlenecks can occur in between the two subsequent calibration steps. The first step is direction independent and is embarrassingly parallel, by distributing the different subbands of a single observation (typically 244) over the different nodes, with one subband per node. Processing can start as soon as a subband has been copied to a node disk. This takes typically four hours, but the next step is direction dependent calibration and its algorithm needs a unified memory space to compute the calibration solutions. This typically takes four days on a single fat node with hundreds of GB of RAM, which would render the remaining nodes idle when processing a single observation. Processing of multiple LOFAR archival observations simultaneously by many containers will reduce latency on the compute nodes after the first calibration step of the first observation has been completed. Also, it is important that reservation of the compute nodes is done in an intelligent manner, i.e., that the nodes will not be waiting for data to arrive at the cluster.

PROCESS can offer access to the three sites where the LOFAR Long Term Archive is stored – Amsterdam, Jülich and Poznan, making simultaneous combined staging and processing of observations possible. Presently, processing of data from different sites requires separate user interfaces. The services required for the LOFAR Use Case pipeline are shown in Figure 7.

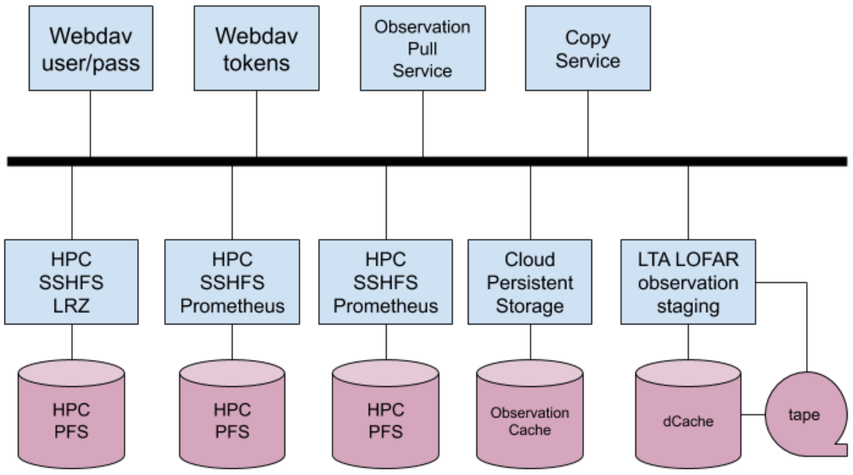


Figure 7. UC#2: Data infrastructure including data adapters as well as long-term-archive staging service

5.3 Experiments

Benchmarks have been performed to assess the performance, mainly in terms of compute and/or transfer duration, of the PROCESS infrastructure and use-case specific services.

5.3.1 Process Data Service in Medical Use Case

The data staging and pre-processing workflow of the medical use case described in Section 5.1 becomes crucial for the performance of the application when the data processing is distributed across geographically distributed data centers. PROCESS data infrastructure has been used to enable the pre-processing workflow at the AGH datacenter in Krakow while the neural network training workflow is at the UvA in Amsterdam.

Table 6 shows the cross site staging part of the Camelyon16 dataset using several protocols and strategies. Our initial approach is to use the widely available SCP protocol and use cluster head nodes to stage the data. From the table, this approach is shown to be one of the worst strategies, e.g. LISA to AGH. Furthermore, head nodes are very unstable for staging such data with frequent stalls and broken connections. A second strategy is to use Data Transfer Nodes (DTN). What we can show from the tests is that using such DTN nodes can speedup transfers, for example, transferring data from LRZ site to LISA is faster by using a DTN relay than by a direct copy. E.g. LRZ-VDTN to AMS-DTN takes 5.85 minutes plus AMS-DTN to LISA is 3.03 minutes, cumulatively it is 8.9 minutes which is $\sim 30\,\%$ faster than a direct copy which is 12.96 minutes. The results also show the added speedup by using campus networks. The AMS-DTN and LISA are on campus thus their bandwidth is approximately 4 times better than the rest. Using DTNs as cache staging nodes could potentially accelerate data transfers between sites. With these tests we feel that we have a basis for the upcoming steps for UC1. Moreover, the additional requirement of SCP protocol for data transfer as stated in D4.3 page 9 has been met. This ensures that data transfer can be applied to different types of users, and especially hospital institutions which may not have open FTP access for security reasons.

Protocol – SCP	30 GB Camelyon16.partAA [MB/s]	30 GB Camelyon16.partAB [MB/s]	30 GB Camelyon16.partAC [MB/s]	Mean BW [MB/s]	Duration [minutes]
LRZ-V-DTN to AMS-DTN	86.8	90.8	84.9	87.5	5.85
LRZ-V-DTN to AGH	33.1	31.2	32.2	32.17	15.92
LRZ-V-DTN to LISA	25.5	64	29	39.5	12.96
AMS-DTN to LISA	167.9	172.6	166	168.83	3.03
AMS-DTN to AGH	53	53.9	38.9	48.6	10.53
LISA to AGH	40	21.3	29.5	30.27	16.91

Table 6. Data transfer tests between different storage locations

5.3.2 Data Services in the LOFAR Use Case

As mentioned in section 5.2, the LOFAR use-case relies on archival observations. To access this data, it has to be staged first. During this process a tape robot will retrieve the appropriate tape(s) and read the desired data to a cache. Thereafter, the data can be accessed directly from the cache. To gain insight into the overhead this introduces to the pipeline, we performed the following three benchmarks: estimating queuing and preparation time; total staging time as a function of total size; transfer speeds from the three LTA locations to each of the three PROCESS HPC clusters; and execution time as a function of total size.

Estimating Queuing and Preparation Time. The tape drives and robots at each of the LTA locations are not only shared by other LTA users, but also by other projects housed in the same data center [29]. Therefore, it may be that a staging request will spend some time in a queue. In addition, the tape robot may need some preparation time before the data can be copied. To estimate this overhead, we staged a small file (< 100 MB) so the reading time would be negligible compared to the remaining queuing and preparation time. We repeated this experiment ten times, at each LTA location, spread over different days, before averaging the recorded durations (Figure 8).

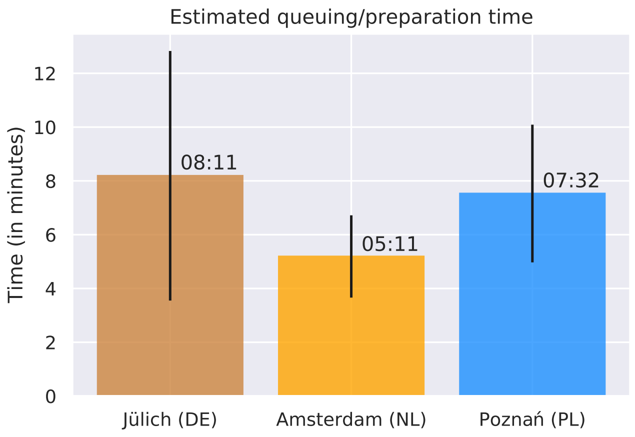


Figure 8. Estimated queuing/preparation time

The durations are quite variable, but in general can be placed in the five to ten-minute range. It might occur that queuing and preparation take longer, especially if multiple staging requests are filled simultaneously. However, we did not experience this during our benchmarking period. The differences between the three locations can be attributed to (possibly) different configurations and/or load at the respective data centers.

Total Staging Time as a Function of Total Size. For an indication about the total duration of a staging request, we staged increasingly large numbers of gigabytes (from 20 GB up to 320 GB). We repeated this three times, at each LTA location, spread over different days before averaging the durations (Figure 9). We observe, again, that the durations are variable, but are reasonable. These durations are uncontrollable, as explained before, because of the shared nature of the LTA systems.

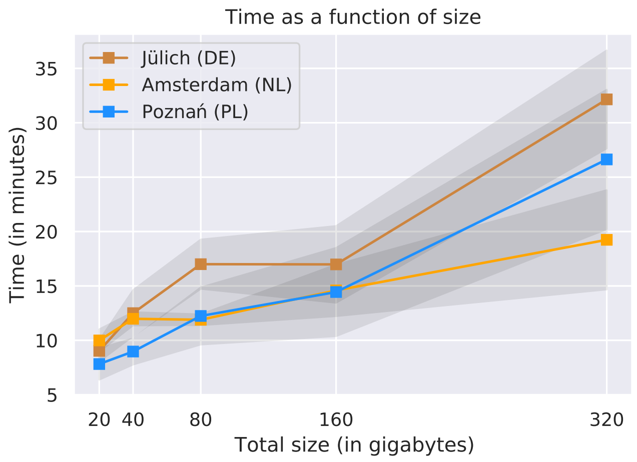


Figure 9. Staging time as a function of size

Transfer Speeds from LTA to HPC Clusters. After the data have been staged, they can be transferred to a HPC cluster for further processing. We are benchmarking this transfer to see to which extent this transfer of large data files across system boundaries induces a bottleneck in the overall use-case's pipeline. We measured the transfer speed (Figure 10) four times before averaging it, between each LTA location and the HPC clusters: LRZ in Garching (DE), LISA in Amsterdam (NL) and CYF in Krakow (PL). Transfer consisted of up to 10 files with a total size of 100 GB.

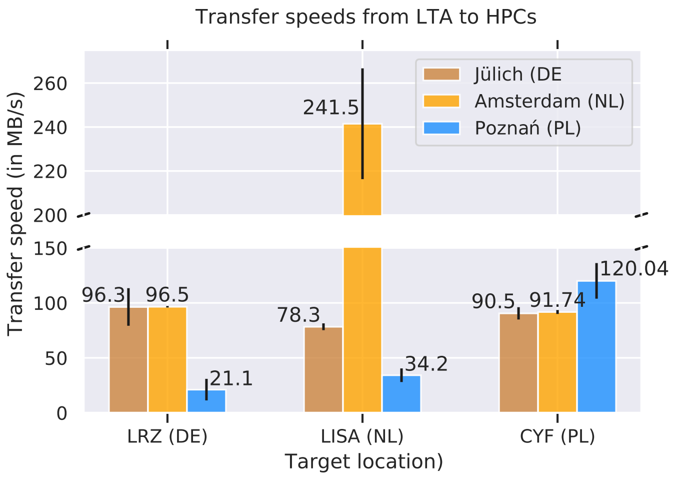


Figure 10. Transfer speeds from LTA to HPCs

The results show that the transfers between LTA locations and HPC clusters are roughly in the 80 to 120 MB/s range. An exception are the transfers between LISA and the Amsterdam LTA location, where the range is significantly higher. This is excepted, since both locations operate on the same optimized grid infrastructure [30]. Another exception is the transfer from the Poznan LTA location to the LRZ and LISA HCP clusters. Although by no means slow, the relative lower speeds may be due to the geographical distance and/or the specific public network composition in place. The achieved speeds between the other locations are, although close to saturating the used public network, still considered suboptimal for exascale purposes. Optimized networks, e.g. with tuned data transfer nodes (DTNs), are needed. Provided with such, the PROCESS infrastructure is able to utilize the improved capabilities and scale along with the transfer speeds, as demonstrated with the transfers between the Amsterdam LTA location and the LISA HCP cluster.

Execution Time as a Function of Total Size (from D3.3). For the LOFAR use-case, we measured the execution time as the most intensive components capable of generating high FLOPS values are currently sequential or multi-threaded. The wall clock times of the main steps of the data reduction pipeline are given in Table 7. The main conclusion to draw from the high magnitude of these values is that the overhead due to scheduling and staging is negligible.

Step	Data Size	Execution Time
1. Calibrator DI	25 GB	~ 2.5 hour
2. Target DI	433 GB	~ 3.5 hour
3. Init-subtract	76 GB	~ 10 hour
4. DD2 (FACTOR)	76 GB	~ 5 days

Table 7. Wall clock times of the main steps of the data reduction pipeline

6 CONCLUSIONS AND FUTURE WORK

In this paper, we present the PROCESS data infrastructure which follows a micro-infrastructure approach. The micro-infrastructure is created at runtime and composed of a set of data service containers instantiated by the core of the infrastructure, LOBCDER. All the interactions among the software components composing the micro-infrastructure have been clearly defined and used to create the implementations of PROCESS application scenarios' data handling pipelines. Throughout the five application use cases, a single backbone data infrastructure has been used and combined at runtime with multiple data services required by the five applications. In this paper we focus on two out of the five applications, more details about the other applications can be found in [53]. While the two applications described in this paper are coming from two different scientific domains, namely astronomy and medical imaging, they share a number of storage and logical adapters, but also have their specific ones. Because of the container centric approach followed in the current implementation of the micro-infrastructure, it was straightforward to mix and match various data adapters to fulfill the applications data handling requirements. All software components developed and reported in this paper are available in the PROCESS software repository [56].

The micro-architecture is currently extended with two software layers to support both application developers and end-users. The developer layer offers the application developer an easy way to implement data processing functions that can be combined to enable a specific data processing pipeline. The user layer offers basic programming constructs, i.e. variables, conditionals, and loops. The user layer is designed for scientists with limited programming experiences, the programming statements have sentence-like structure [31].

The performance results reported in this paper measure only the overhead induced by the software services proposed in the PROCESS project. In the absence of an exascale computer, exascale can be achieved by combining the power of geographically distributed data centers. However, using the standard data transfer techniques the stage-in of the 7 PB of data currently in LOFAR LTA would take about 7 years and transferring a full exabyte would take several centuries. On-going research to address this problem proposes to use efficient DTN networks between compute centers involved in large data transfers across these centers. Experiments by Geant in 2018 [32] have shown that 100 GbE DTN networks are feasible on the European and even global scale. At such transfer rates it would take less than a minute to transfer a single 16 TB LOFAR observation.

Acknowledgment

This work is supported by the "PROviding Computing solutions for ExaScale ChallengeS" (PROCESS) project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 777533, by the project APVV-17-0619 (U-COMP) "Urgent Computing for Exascale Data" and by the VEGA project "New Methods and Approaches for Distributed Scalable Computing" No. 2/0125/20.

REFERENCES

[1] Koehler, M.—Knight, R.—Benkner, S.—Kaniovskyi, Y.—Wood, S.: The VPH-Share Data Management Platform: Enabling Collaborative Data Management for the Virtual Physiological Human Community. 2012 Eighth International Conference on Semantics, Knowledge and Grids, Beijing, China, 2012, pp. 80–87, doi: 10.1109/SKG.2012.51.

[2] Bent, J.—Faibish, S.—Ahrens, J.—Grider, G.—Patchett, J.—Tzelnic, P.—Woodring, J.: Jitter-Free Co-Processing on a Prototype Exascale

Storage Stack. 2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), San Diego, CA, USA, 2012, pp. 1–5, doi: 10.1109/MSST.2012.6232382.

[3] Filippidis, C.: Parallel Storage Systems for Large Scale Machines. http://sc16.supercomputing.org/sc-archive/doctoral_showcase/doc_files/drs104s2-file2.pdf.

[4] Lofstead, J.—Jimenez, I.—Maltzahn, C.—Koziol, Q.—Bent, J.—Barton, E.: DAOS and Friends: A Proposal for an Exascale Storage System. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '16), Salt Lake City, UT, USA, IEEE, 2016, pp. 585–596, doi: 10.1109/SC.2016.49.

[5] Hemsoth, N.: Exascale Storage Gets a GPU Boost DFAF. 2018, https://www.nextplatform.com/2018/02/12/exascale-storage-gets-gpu-boost/.

[6] Hemsoth, N.: An Exascale Timeline for Storage and I/O System. 2017, https://www.nextplatform.com/2017/08/16/exascale-timeline-storage-io-systems/.

[7] Infinite Memory Engine: The Exascale-Era Storage Architecture. 2017, https://www.hpcwire.com/2017/08/21/infinite-memory-engine-exascale-era-storage-architecture/.

[8] Hick, J.—Watson, D.—Cook, D.—Minton, J.—Newman, H.—Preston, T.—Rich, G.—Scott, C.—Shoopman, J.—Noe, J.—O'Connell, J.—Shipman, G.—White, V.: HPSS in the Extreme Scale Era: Report to DOE Office of Science on HPSS in 2018–2022. Technical Report No. LBNL-3877E, Lawrence Berkeley National Lab. (LBNL), Berkeley, CA, USA, 2009.

[9] Cope, J.—Liu, N.—Lang, S.—Carns, P.—Carothers, C.—Ross, R.: CODES: Enabling Co-Design of Multi-Layer Exascale Storage Architectures. https://pdfs.semanticscholar.org/159d/bd0a8c18e2df895b131e33499e2d529210e0.pdf.

[10] Mair, J.—Huang, Z.—Eyers, D.—Chen, Y.: Quantifying the Energy Efficiency Challenges of Achieving Exascale Computing. 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, China, 2015, pp. 943–950, doi: 10.1109/CCGrid.2015.130.

[11] Cameron, K. W.: Energy Efficiency in the Wild: Why Datacenters Fear Power Management. Computer, Vol. 47, 2014, No. 11, pp. 89–92, doi: 10.1109/MC.2014.315.

[12] https://www.eudat.eu/services/b2stage.

[13] https://www.dcache.org.

[14] https://www.gluster.org.

[15] https://iRODS.org.

[16] https://www.dcache.org/manuals/2016/presentations/20161006-PM-dCache.pdf.

[17] Shannigrahi, S.—Fan, C.—Papadopoulos, C.: Named Data Networking Strategies for Improving Large Scientific Data Transfers. Proceedings of the IEEE International Conference on Communications Workshops (ICC Workshops), Kansas City, MO, USA, 2018, doi: 10.1109/ICCW.2018.8403576.

[18] Chen, S.—Cao, J.—Zhu, L.: NDSS: A Named Data Storage System. 2015 International Conference on Cloud and Autonomic Computing, Boston, MA, USA, 2015, pp. 196–199, doi: 10.1109/ICCAC.2015.12.

[19] Zhu, S.—Yuan, M.—Lei, K.: Ndynamo: An ndnDHT-Based Distributed Storage System over Named Data Networking. 2016 5th International Conference on Computer Science and Network Technology (ICCSNT), Changchun, China, 2016, pp. 148–152, doi: 10.1109/ICCSNT.2016.8070137.

[20] Rao, Y.—Gao, D.—Zhang, H.—Foh, C. H.: Mobility Support for the User in NDN-Based Cloud Storage Service. 2015 IEEE Globecom Workshops (GC Wkshps), San Diego, CA, USA, 2015, pp. 1–6, doi: 10.1109/GLOCOMW.2015.7414159.

[21] Cloudify Orchestration Service. `https://docs.cloudify.co/latest/developer/apis/`.

[22] Atkinson, M. P.—Galea, M.—Liew, C. S.—Martin, P.: ADMIRE – Final Report on the ADMIRE Architecture, with an Assessment and Proposals for Its Development. Technical Report, The ADMIRE Project, May 2011.

[23] Brezany, P.—Aranda, C. B.—Corcho, O.—Janciak, I.—Woehrer, A.—Atkinson M.: ADMIRE – Report Defining the Final Iteration of the Model and Language. Deliverable Report D1.9, The ADMIRE Project, May 2011.

[24] Available at GitHub: `https://github.com/micro-infrastructure/adaptor-sshfs`.

[25] Available at GitHub: `https://github.com/micro-infrastructure/service-webdavserver`.

[26] Available at GitHub: `https://github.com/micro-infrastructure/service-scp2scp`.

[27] Available at GitHub: `https://github.com/micro-infrastructure/service-nextcloud`.

[28] Graziani, M.—Eggel, I.—Deligand, F.—Bobák, M.—Andrearczyk, V.—Müller, H.: Breast Histopathology with High-Performance Computing and Deep Learning. Computing and Informatics, Vol. 39, 2020, No. 4, pp. 780–807, doi: 10.31577/cai_2020_4_780.

[29] `https://www.astron.nl/lofarwiki/doku.php?id=start`.

[30] `https://www.surf.nl/en/use-case-space-research-with-grid-infrastructure`.

[31] `https://onnovalkering.github.io/brane/`.

[32] `https://github.com/recap/MicroInfrastructure`.

[33] TOP500.org. The TOP500 List (June 2008). 2008, `https://www.top500.org/lists/2008/06/`. Accessed: 04-01-2018.

[34] Bándi, P.—Geesing, O.—Manson, Q.—Van Dijk, M.—Balkenhol, M.: From Detection of Individual Metastases to Classification of Lymph Node Status at the Patient Level: The CAMELYON17 Challenge. IEEE Transactions on Medical Imaging, Vol. 38, 2019, No. 2, pp. 550–560, doi: 10.1109/TMI.2018.2867350.

[35] Vijayaraghavan, T.—Eckert, Y.—Loh, G. H.—Schulte, M. J.—Ignatowski, M.—Beckmann, B. M.—Brantley, W. C.—Greathouse, J. L.—Huang, W.—Karunanithi, A. et al.: Design and Analysis

of an APU for Exascale Computing. 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), IEEE, 2017, pp. 85–96, doi: 10.1109/HPCA.2017.42.

[36] HGST Ultrastar Hs14. `http://www.hgst.com/products/harddrives/ultrastar-hs14`. Accessed: 03-12-2017.

[37] ASTC Technology Roadmap. `http://idema.org/?pageid=5868`. Accessed: 03-12-2017.

[38] Sandberg, R.—Goldberg, D.—Kleiman, S.—Walsh, D.—Lyon, B.: Design and Implementation or the Sun Network File System. Proceedings of the Summer 1985 USENIX Conference, Portland, OR, USA, 1985, pp. 119–130.

[39] Levy, E.—Silberschatz, A.: Distributed File Systems: Concepts and Examples. ACM Computing Surveys (CSUR), Vol. 22, 1990, No. 4, pp. 321–374, doi: 10.1145/98163.98169.

[40] Roselli, D. S.—Lorch, J. R.—Anderson, T. E.: A Comparison of File System Workloads. Proceedings of the USENIX Annual Technical Conference, General Track, 2000, pp. 41–54.

[41] Niazi, S.—Ismail, M.—Haridi, S.—Dowling, J.—Grohsschmiedt, S.—Ronström, M.: HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases. Proceedings of the 15th Usenix Conference on File and Storage Technologies (FAST 2017), Santa Clara, CA, USA, pp. 89–103.

[42] Takatsu, F.—Hiraga, K.—Tatebe, O.: PPFS: A Scale-Out Distributed File System for Post-Petascale Systems. Journal of Information Processing, Vol. 25, 2017, pp. 438–447, doi: 10.2197/ipsjjip.25.438.

[43] Ren, K.—Zheng, Q.—Patil, S.—Gibson, G.: IndexFS: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14), IEEE, 2014, pp. 237–248, doi: 10.1109/SC.2014.25.

[44] Benet, J.: IPFS – Content Addressed, Versioned, P2P File System. arXiv preprint arXiv:1407.3561, 2014.

[45] Yahoo Cloud Object Store – Object Storage at Exabyte Scale. `https://yahooeng.tumblr.com/post/116391291701/yahoocloud-object-store-object-storage-at`. Accessed: 06-01-2018.

[46] Koulouzis, S.—Belloum, A. S. Z.—Bubak, M. T.—Zhao, Z.—Živković, M.—de Laat, C. T. A. M.: SDN-Aware Federation of Distributed Data. Future Generation Computer Systems, Vol. 56, 2016, pp. 64–76, doi: 10.1016/j.future.2015.09.032.

[47] Koulouzis, S.—Belloum, A.—Bubak, M.—Lamata, P.—Nolte, D.—Vasyunin, D.—de Laat, C.: Distributed Data Management Service for VPH Applications. IEEE Internet Computing, Vol. 20, 2016, No. 2, pp. 34–41, doi: 10.1109/MIC.2015.71.

[48] `https://www.nist.gov/publications/nist-big-data-interoperability-framework-volume-1-definitions`.

[49] `https://lta.lofar.eu`.

[50] https://www.process-project.eu/wp-content/uploads/2020/02/PUBLIC_PROCESS_D4.1_Initial_-state_of_the_art_and_requirement_analysis_initial_PROCESS_architecture_v1-1.pdf.

[51] https://www.process-project.eu/wp-content/uploads/2020/02/PUBLIC_PROCESS_D4.2_Report_on_architecture_v1.0.pdf.

[52] https://www.process-project.eu/wp-content/uploads/2020/02/PUBLIC_PROCESS_D5.1_Design_of_data_infrastructure_v1.0.pdf.

[53] https://www.process-project.eu/wp-content/uploads/2020/02/PROCESS_D5.2_Alpha_release_of_the_Data_service_v1.0.pdf.

[54] Spreeuw, H.—Madougou, S.—Van Haren, R.—Weel, B.—Belloum, A.—Maassen, J.: Unlocking the LOFAR LTA. 2019 15th International Conference on eScience (eScience), San Diego, CA, USA, 2019, pp. 467–470, doi: 10.1109/eScience.2019.00061.

[55] https://www.process-project.eu/wp-content/uploads/2020/02/PUBLIC_PROCESS_D2.1_Progress_Report_v1.0.pdf.

[56] https://www.research-software.nl/.

[57] Wittenburg, P.—Van de Sompel, H.—Vigen, J.—Bachem, A.—Romary, L.—Marinucci, M.—Andersson, T.—Genova, F.—Best, C.—Los, W. et al.: Riding the Wave: How Europe Can Gain from the Rising Tide of Scientific Data. Final Report of the High Level Expert Group on Scientific Data – A Submission to the European Commission, October 2010. http://ec.europa.eu/newsroom/dae/document.cfm?doc_id=707.

Reginald Cushing is PostDoc at the University of Amsterdam in the Multiscale Networked Systems (MNS) group. His research fields are in distributed systems with a focus on data processing, federation, and scientific workflows.

Onno Valkering is Scientific Programmer in the Multiscale Networked Systems (MNS) research group at the University of Amsterdam, Holland. His interests are distributed data processing, domain-specific languages, and privacy-preserving techniques.

Adam Belloum is Senior Researcher at the Computer Science Department of the University of Amsterdam and the technology lead working on optimized data handling at the Dutch National eScience Center. He received his M.Sc. and Ph.D. degrees from the Compiegne University of Technology, France.

Souley Madougou is an eScience engineer at the Netherlands eScience Centre since December 2018. He is mainly involved in the PROCESS project in which he contributes to the implementation of the LOFAR use case and the development and analysis of PROCESS performance models. He previously worked in several eScience projects in the Netherlands. His research interests include performance modelling on many-core architectures, parallel programming and provenance.

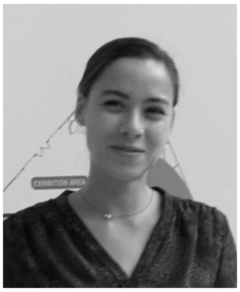
Martin Bobak is Scientist at the Institute of Informatics, Slovak Academy of Sciences, Bratislava, Slovakia, in the Department of Parallel and Distributed Information Processing. He started working at the institute in 2013, defended his dissertation thesis at the institute in 2017, became Member of the Scientific Board of the institute, and Guest Handling Editor in the CC Journal Computing and Informatics. His field of research is cloud computing and the architectures of distributed cloud-based applications. He is the author of numerous scientific publications and has participated in several European and Slovak R & D projects.

Ondrej Habala is Researcher at the Institute of Informatics, Slovak Academy of Sciences, Bratislava, Slovakia. He works in the Department of Parallel and Distributed Information Processing since 2001. His interests are mainly in data and metadata management in distributed computing, as well as in distributed information systems in general and focused on applications in environmental sciences and hydro-meteorology. He has over the years participated in numerous FP5, FP6, FP7, H2020 and national research projects and produced over 80 scientific publications.

Viet Tran is Senior Researcher at the Institute of Informatics, Slovak Academy of Sciences (IISAS). His primary research fields are complex distributed information processing, grid and cloud computing, system deployment and security. He received M.Sc. degree in informatics and information technology, Ph.D. degree in applied informatics from the Slovak University of Technology (STU) in Bratislava, Slovakia. He actively participates on preparations and solving a number of EU IST RTD 4th, 5th, 6th, 7th FP and EU H2020 projects such as PROCESS, DEEP-HybridDataCloud, EOSC-Hub and EOSC-Synergy. He is the author or co-author of over 100 scientific publications.

Jan Meizner has graduated majoring in federated IT security systems. Since then he has been working at ACC Cyfronet AGH on many EU and national projects involving a wide range of subjects, including computational medicine. His work focuses on IT security, operations of cloud and HPC infrastructures, as well as building software for such infrastructures. Currently involved also in Sano Centre for Computational Medicine, focusing on the operations of IT systems, as well as a range of IT security tasks, including identity management and data security.

Piotr Nowakowski is Research Programmer at the Academic Computing Centre CYFRONET AGH and Senior Data Scientist at the Sano Centre for Computational Medicine. He specializes in design and development of distributed environments for computational science, and he has participated in a range of national and international research initiatives, including EU-funded projects – most recently VPH-Share, EurValve and PROCESS. He is the author or co-author of over 100 scientific publications.



Mara Graziani is a third-year Ph.D. student with double affiliation at the University of Geneva and at the University of Applied Sciences of Western Switzerland. With her research, she aims at improving the interpretability of machine learning systems for healthcare by a human-centric approach. She was a visiting student at the Martinos Center, part of Harvard Medical School in Boston, MA, USA to analyze the interaction between clinicians and deep learning systems. From her background of IT Engineering, she was awarded the Engineering Department Award for completing the M.Phil. in machine learning, speech and language at the University of Cambridge, UK in 2017.

Henning Müller is Full Professor at the HES-SO Valais and responsible for the eHealth unit of the school. He is also Professor at the Medical Faculty of the University of Geneva and has been on sabbatical at the Martinos Center, part of Harvard Medical School in Boston, MA, USA to focus on research activities. He is the coordinator of the ExaMode EU project, was the coordinator of the Khresmoi EU project, the scientific coordinator of the VISCERAL EU project, and is the initiator of the ImageCLEF benchmark that has run medical tasks since 2004. He has authored over 500 scientific papers with more than 13 000 citations and is in the editorial board of several journals.