This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 777533.

PROviding Computing solutions for ExaScale ChallengeS

D3.3	Performance modelling and prediction – final approach		
Project:	PROCESS H2020 – 777533	**Start / Duration:**	01 November 2017 36 Months
Dissemination[1]:	PU	**Nature[2]:**	**R**
Due Date:	30 April 2020	**Work Package:**	**WP 3**
Filename[3]	PROCESS_D3.3_PerformanceModellingAndPrediction-FinalApproach_v1.0.docx		

ABSTRACT

This deliverable is an update of D3.2 and based on its content. It finalizes the performance modelling and prediction approaches outlined in D3.2 based on the results obtained during the project.
It describes the performance modelling and influences thus the design, development and validation of the components of the PROCESS infrastructure.

1 PU = Public; CO = Confidential, only for members of the Consortium (including the EC services).

2 R = Report; R+O = Report plus Other. Note: all "O" deliverables must be accompanied by a deliverable report.

3 eg DX.Y_name to the deliverable_v0xx. v1 corresponds to the final release submitted to the EC.

Deliverable Contributors:	Name	Organisation	Role / Title
Deliverable Leader[4]	Pancake-Steeg, Jörg	LSY	Coordinator
Contributing Authors[5]	Madougou, Souley; Maassen, Jason	NLESC	Writers
	Valkering, Onno; Cushing, Reggie	UvA	Writers
	Graziani, Mara	HES-SO	Writer
	Schmidt, Jan; Höb, Maximilian	LMU	Writers
	Meizner, Jan	AGH	Writer
	Somoskői, Balázs; Matejko, Pawel	LSY	Writers
Reviewer(s)[6]	Habala, Ondrej	UISAV	Reviewer
	Belloum, Adam	UvA	Reviewer
Final review and approval	Höb, Maximilian	LMU	Coordinator

Document History

Release	Date	Reasons for Change	Status[7]	Distribution
0.0	31.10.2019	Final Version of D3.2	Draft	Working Group
0.1	15.01.2020	Use Case updates	Draft	Working Group
0.2	15.05.2020	Use Case updates	Draft	Working Group
0.3	29.05.2020	Measurements update	Draft	Consortium
0.4	05.06.2020	Evaluation update	Draft	Consortium
0.5	17.06.2020	Internal review	In Review	Consortium
0.6	22.06.2020	Final internal review	In Review	Consortium
1.0	30.06.2020	Release	Released	Public

4 Person from the lead beneficiary that is responsible for the deliverable.

5 Person(s) from contributing partners for the deliverable.

6 Typically, person(s) with appropriate expertise to assess the deliverable quality.

7 Status = “Draft”; “In Review”; “Released”.

Table of Contents

Executive Summary

This document presents the foundations of the performance modelling and prediction approaches that the PROCESS project will use to steer its design, development and validation efforts. The broad range of environments that the PROCESS software will run on presents obvious challenges in the development of a uniform, easy-to-use and straightforward performance model. The necessary streamlining and simplification of the approach should not omit any relevant aspects that are determining the actual performance as observed by a user.

As a way to balance these conflicting needs, the project will use a solution based on measurable performance metrics, complemented by a mathematical model that allows extrapolating performance on systems that are considerably more complex than the current ones. The extrapolation will also be necessary to understand the impact of advances in the capacities of individual components will have in the execution speed of complex workflows.

The model used by the project assumes that typical exascale applications can be modelled as pipelines consisting of the input data stage-in, processing and (result) data stage-out steps. However, for workflows comprising several dynamically configured and deployed components, the set of performance components need to be able to analyse the execution in a more fine-grained manner. The full set of metrics consists of:

- T1: Configuration of the workflow
- T2: Deployment strategy (selection of resources)
- T3: Stage-in of the data
- T4: Container selection (fetching the container encompassing the executable code, as defined in T1)
- T5: Scheduling (time spent on the queue of a compute system)
- T6: Execution time
- T7: Stage-Out Strategy (choosing the approach based on required storage capacity, type and availability)
- T8: Stage-Out (actual transfer of data).

It should be noted that some of these steps depend on user input, therefore, the overall execution time will depend on the expertise and skills of the user. There are also considerable differences between situations where all the necessary resources belong to a single system, on multiple platforms controlled by a single organisation or in a federated system crossing organisational and geographical boundaries.

To focus performance-related development efforts, the PROCESS performance model groups the metrics into the following categories:

$$Overhead = T1 + T2 + T4 + T7$$

$$Data\ Transfer = T3 + T8$$

$$Scheduling = T5, \quad Execution\ Time = T6$$

The overhead consists of factors that can be influenced by the PROCESS software, while the data transfer and execution time components are primarily dependent on the performance of the networking and computing hardware available. The scheduling is highly dependent on the number of competing jobs and the policies (e.g. priority queue available for the job). However, similar to the characteristics of the underlying hardware, scheduling is an issue that can't be influenced by the design of the software.

As the relative impact of these four categories on the system-level performance as experienced by the user can vary dramatically, the project will develop a user-configurable workflow that will be used to complement actual use case software in the evaluation of the PROCESS platform. However, it should be noted that the use cases already stress the different aspects of the equation in a quite comprehensive manner. For example, UC1 performance will be highly dependent on the data transfer and execution time components, whereas the interactive use anticipated in the UC4 will require minimising all of the overheads in the PROCESS platform.

List of Figures

List of Tables

1 Introduction

This deliverable D3.3 updates the approach to model the performance of the PROCESS infrastructure and its possible scalability towards exascale workflows. Based on D3.1 and D3.2 this deliverable D3.3 enhances and completes the process of developing a performance model. It gives the opportunity to provide predictions of the architecture behaviour towards extreme large workflow executions.

In order to achieve exascale performance, we need on the one hand local computing centres capable of running at such an exascale level. On the other hand, one also needs software being deployable not only across several nodes, but also across different locations across Europe, the so-called sites. For our use cases presented in earlier and related deliverables and based on PROCESS's architectural design decision, we consider this a prerequisite. In order to technically facilitate the decision, we seek for the approach to containerize the architectural elements as well as to push all use cases to design their execution in containers. This will allow for deploying instances of independent executions on subsets of a given data set on different local nodes and at the same time on different geographical based sites.

However, the hardware and the software development towards exascale is an ongoing process and we have to face the challenge to predict a behaviour that cannot be verified within the lifetime of this project. Therefore, we had to develop a prediction model based on measurable performance indicators and from there on extrapolating runtime behaviour towards a much higher scale. The model needs to meet the requirements to predict the behaviour of all our services and the PROCESS infrastructure as a whole but must also be able to adapt new requirements coming from future and new applications.

To distinguish the most common approaches for performance prediction models, we will first give an overview and classification of up-to-date performance modelling and prediction methods, on the basis of which we will present the approach of choice for PROCESS.

1.1 Performance modelling approaches

Performance modelling is used for many computational and storage systems around Europe. Regarding the exascale challenge, also other EU projects examine the needs and conclusions to enable exascale performance.

The CRESTA[8] project (Collaborative Research Into Exascale Systemware, Tools and Applications) proposes a framework focusing on software and tool developments for end-user scientist. Their solution is limited to local site needs and deals mainly with hardware decisions owners of supercomputing centres will face in the next years.

1.1.1 Overview and classification

One of the CRESTA project partners is David Henty from the Edinburgh Parallel Computing Centre (EPCC). In his publications he gives an overview on generic performance modelling techniques and a classification of which. In Table 1 he defines four main categories varying from raw measurements, over benchmarking and simulations to complex analytical modelling with a large number of parameters.

[8] https://www.cresta-project.eu

Technique	Description	Purpose
Measurement	running full applications under various configurations	determine how well application performs
Microbenchmarking	measuring performance of primitive components of application	provide insight into application performance
Simulation	running application or benchmark on software simulation	examine “what if” scenarios e.g. configuration changes
Analytical Modelling	devising parameterized, mathematical model that represents the performance of an application in terms of the performance of processors, nodes, and networks	rapidly predict the expected performance of an application on existing or hypothetical machines

Table 1: Performance Modelling Approaches, cited from [PMO]

Any of the techniques mentioned above will be useful within the PROCESS project:

Measurement
Both simple measurements as well as complex model measurement values are the basis of success. In Section 2, we will define at which points of the execution sequence meaningful measurements can be taken. Measurement values are to deliver input data for further modelling and prediction steps.

Microbenchmarking
Microbenchmarking is used to identify performance bottlenecks in the PROCESS architecture and assists in debugging and verifying its correctness. The microbenchmark is a very simple application (validation or test pipeline) running through the complete PROCESS architecture and gathering first results.

Simulation and Analytical Modelling
Executing and measuring a given application running on PROCESS in different configurations and settings forms the input dataset for this step. The goal of this step is to extrapolate the behaviour and runtime of the application from the given observations. The resulting model will allow for predictions of runtime behaviour beyond the configuration scales measured, which gives us the chance to forecast the performance on an exascale level.

1.1.2 PROCESS Performance Model

Based on the previous description we choose a measurement-based approach with extrapolation through analytical modelling. First the measurands are identified and measurements are performed. In the next step a microbenchmark to evaluate these measurands is developed. Finally, to predict the performance of PROCESS, we use these results to create an analytical model that will allow us to extrapolate the performance based on given measurements.

2 Identification of Measurands

In the previous Section we categorized the approaches to performance modelling and prediction. One of which was a measurement-based approach with extrapolation for performance prediction. To achieve this goal, it is necessary to identify the appropriate measurands within the PROCESS infrastructure that can be used to model the performance of the infrastructure and predict its scaling.

We stress that the hardware infrastructure such as computing, storage, and network have a big impact on the performance of PROCESS services. However, as a project, we have no real influence on this part of the infrastructure. Therefore, our performance measurands focus on the overhead introduced by the software services, but also measure all other relevant numbers to identify relations between them.

In the absence of true exascale systems, our objective, as stated in Section 1, is to achieve exascale by combining the power of geographically distributed data centres. Unfortunately, the traditional configuration of compute centres is more optimized for inner data transfer rather than for outside transfers. While technical solutions to optimize data-transfers exist such as the Data Transfer Nodes[9,10], implementing those solutions is beyond the scope of the project. In PROCESS we try to hide the data transfers by overlapping data transfer with computing or use pre-fetching and caching to minimize the data transfers.

Based on the five use cases defined in PROCESS, we can think of a typical application as a pipeline of data processes which typically requires a data stage-in step followed with an execution step, and finally a data stage-out step. The time required for stage-in and out is expected to be significant, because of the necessary data movement between data centres.

T1: Configuration

The Interactive Execution Environment provides an end-user web portal, where each run of any application needs to be configured. For the different use cases, these configurations vary as shown in the deliverables D4.2 and D5.2.

T2: Deployment Strategy

Part of T2 is the time needed to decide on which computing site[s] and storage site the containers and their data will be deployed. It also needs to initiate the required micro-architecture.

T3: Stage-In

Impact by the access to data services in data centre. However, if PROCESS can make use of caching, proactive pre-fetching or pre-processing we can reduce the impact of T3 on the overall execution performance

T4: Container selection

The workflow that has been defined in T1 specifies a container that will be executed as well as its version. This version needs to be fetched from the container repository and later deployed as a job in T5.

T5: Scheduling

The time a job spends in the queue of the compute resource. This time can vary and will be hard to predict since it's affected by each compute site's scheduling system that isn't in the scope of PROCESS. We may however be able to estimate an upper bound on the queue waiting time that could be added to the actual runtime prediction.

[9] Building User-friendly Data Transfer Nodes, https://www.delaat.net/posters/pdf/2018-11-12-DTN-SURFnet.pdf

[10] Pacific Research Platform https://bozeman-fiona-workshop.ucsd.edu/materials/20180303-dart-dtn-strategic-asset-v1.pptx/view

T6: Execution time

T6 is the time a job takes from leaving the queue to finishing its calculations on the compute resource. This time is determined by the performance and scalability of the application on the selected compute resource. To predict this time, an application specific performance model is required.

T7: Stage-Out Strategy

After the job is done, it may have generated large amounts of output data that needs to be transferred from the compute resource's scratch space back to the PROCESS storage infrastructure. Based on the amount of data and the specified workflow the data service needs to choose a suitable stage-out strategy.

T8: Stage-Out

With the appropriate stage-out strategy the output data now needs to be transferred to the chosen storage resource.

Figure 1 shows a sequence diagram describing all the steps involved in the execution of an application scenario. For each step we define the time corresponding to its completion as follows:

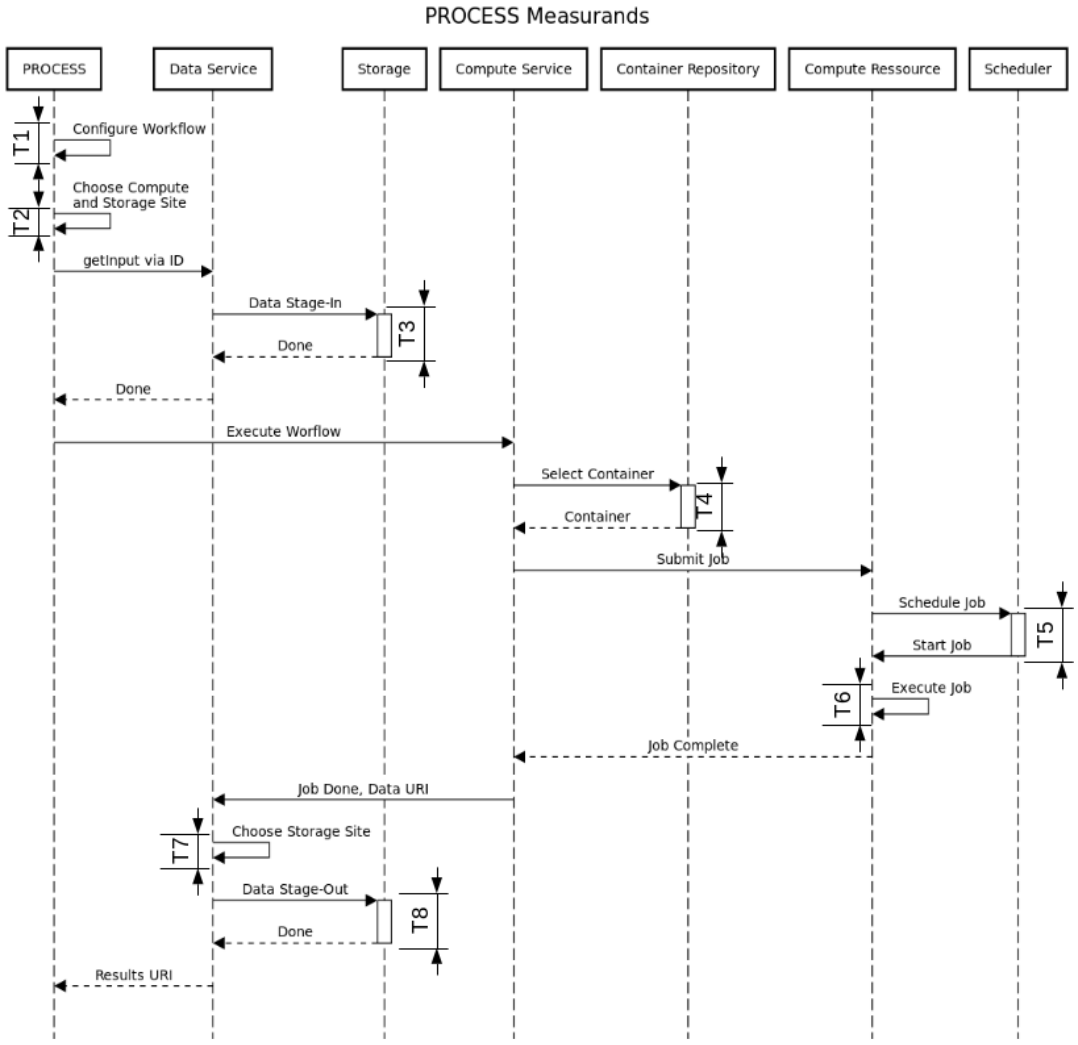

Figure 1: Sequence diagram describing the steps involved in execution of a typical application scenario

Table 2 summarises the various identified times, we will use as performance measurands.

TX	Name	Description
T1	Configuration	Time to configure the workflow for the application
T2	Deployment Strategy	Time to select appropriate storage and computing site
T3	Stage-In	Time to transfer data from source to selected storage site
T4	Container selection	Time to select specified container for the workflow from repository
T5	Schedule	Time the submitted job spends in queue
T6	Execution time	Time spent executing the job on the compute resource
T7	Stage-Out Strategy	Time to select appropriate storage site for output
T8	Stage-Out	Time to transfer result to storage site

Table 2: Description of the PROCESS measurands

Using the identified performance measurands listed in Table 2 we propose a three-step approach to the modelling and performance prediction of the PROCESS infrastructure. **First**, we will show that the overhead of the PROCESS platform for a deployment on one site (initializing the micro-infrastructure and scheduling) is negligible. **Second**, since the deployment strategy of process is to deploy every application containerized, we show the weak scaling capabilities of PROCESS by deploying multiple containers with a split of the input data on one site. And **third**, since the goal is to achieve an exascale system solution, we enable applications to scale by splitting the data and deploying containers across multiple sites of PROCESS.

We therefore describe three measurement scenarios:

Scenario 1: Single container – single site (Figure 3-a)

In this scenario we measure the execution time of processing the input sequentially within one container running. This container uses the maximal possible and available number of compute resources PROCESS can use at one single site (e.g. use case 2 running only at one cluster).

Scenario 2: Multiple containers – single site (Figure 3-b)

In the second scenario we submit several containers on one cluster. Here, we either expect a speedup, since the container in scenario 1 eventually did not fully utilize compute resources or the same runtime as before, since the overhead to deploy more than one container in parallel should be minimal.

Scenario 3: Multiple containers – multiples sites (Figure 3-c)

This last scenario will deploy several containers in parallel on two different sites with an also split input data set. We expect a significant speedup since multiple containers will be deployed on multiple sites.

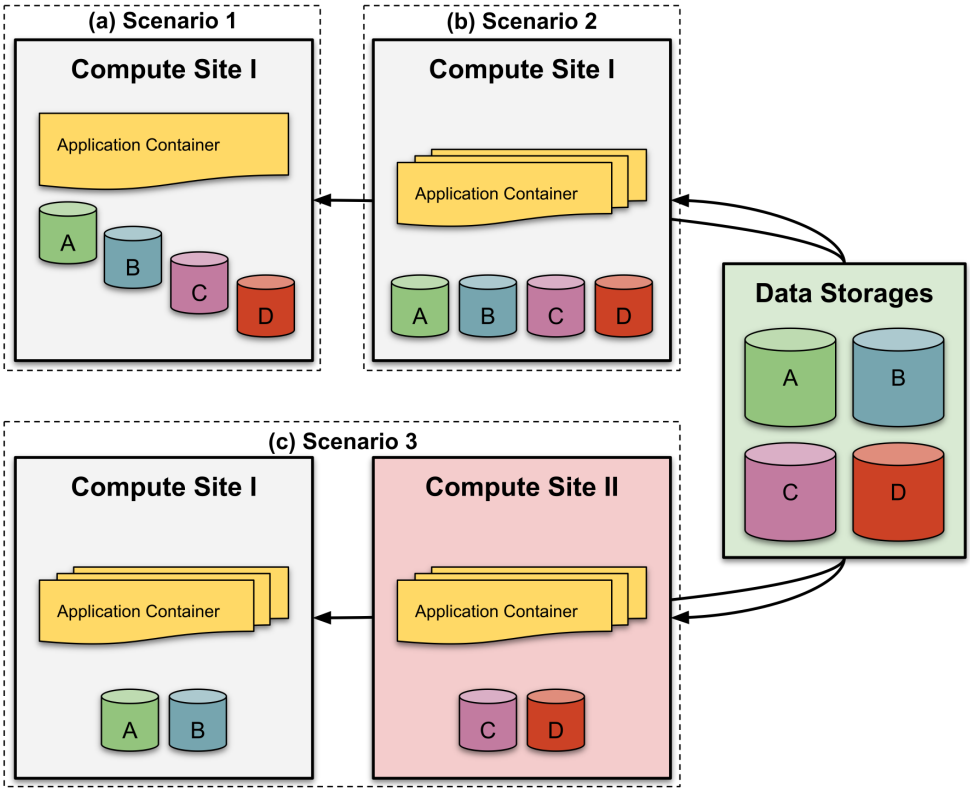

Figure 2: Three measurement scenarios

***Figure 2**: Three measurement scenarios: (a) Single container – single site, (b) Multiple container – single site, (c) Multiple container – multiple site. In all three scenarios Stage-In and Stage-Out will down scale the system overall performances, unless we address the data transfer over a wide area network.*

After evaluating these scenarios and measurements, we will present a generic performance model that allows to predict the scalability of the PROCESS infrastructure for a given application.

3 Development of a balanced Prediction Model

In this section we will present our approach to determine the components of a simple predictive model for workflow performance on the PROCESS infrastructure.

3.1 Runtime Composition

Based on Figure 2 the total runtime of an application can be defined as follows:

$$Runtime = Overhead + Data\ Transfer + Scheduling + \ Execution\ Time$$

Where:

$$Overhead = T1 + T2 + T4 + T7;$$
$$Data\ Transfer = T3 + T8,;$$
$$Scheduling = T5;$$
$$Execution\ Time = T6$$

The *overhead* component contains all overhead directly related to the PROCESS services. This includes selecting the appropriate resources for data access and compute in the Execution Environment, configuring the micro-architecture of LOBCDER for data access, fetching the application containers, and submitting the application to the selected resource using Rimrock.

To support exascale it is important that this *overhead* is low per submitted workflow and does not depend on the scale of the compute resources which are targeted by PROCESS services. We expect that this overhead component is orders of magnitude smaller than the other components and will therefore be negligible.

The *data transfer, scheduling and execution time* components are mostly determined by factors outside of the control of PROCESS services, such as network capacity, queue waiting times, and how well a workflow performs and scales on a given resource. Nevertheless, having an estimate of the *data transfer and scheduling delay* is useful for selecting a resource to which a workflow should be submitted. If *execution time* estimates are available, this selection may be improved further, and a total *runtime* estimate may be provided to the user.

The *data transfer* component is mainly determined by two parts: the time required by Dispel to perform pre-processing of the data (if any), and the time required to transfer the resulting data volume given the end-to-end transfer capacity between the storage and compute site. These two components may largely overlap if the data pre-processing is simple and can be performed on the fly, but for complex operations this may not be the case.

For the latter part, predicting large long-distance data transfers, a significant amount of research has been performed in the last two decades. For example, [[Liu2017]] describes a model that predicts end-to-end data transfer times with high accuracy based on logs of the Globus transfer service. Similarly, much research has been done on estimating queue waiting times of HPC applications which dominates the *scheduling* component. For example, [[Nurmi2007]] describes a model that provides estimates with a high degree of accuracy and correctness for a large number of supercomputing sites.

For PROCESS we will re-use this existing work to provide estimates for both the data transfer and scheduling components of the model.

Predicting the *execution time* is highly application specific and must be done separately for each of the use cases. It may be dependent on input datasets, application parameters, number of resources used (number and type of cores, amount and speed memory, availability and type of GPUs, etc).

Strong scalability of the use case applications is expected to be limited well below exascale, as currently only few applications are able to exploit a petascale level. To determine the limits of the strong scalability of the use case workflows, traditional performance benchmarking of the applications can be used for representative input data sets and parameters. To circumvent limits in strong scalability, we can exploit weak scalability, where multiple workflows are running at the same time to process different datasets. However, doing so may shift the bottleneck from the application to other

sources, such as the data service, or local storage on the resources. Such limits can be discovered by performing weak scalability testing, both on a single site and multiple sites.

Unfortunately, it requires a large effort to create a complete and accurate model of the application behaviour for each of the use cases. Although users may be willing to perform some testing in advance to tune their application, they are mostly interested in obtaining application results. Therefore, highly accurate modelling of the application workflows is not required, instead a rough estimate of the processing time is generally enough.

We will initially assume the user will provide an estimate for the execution time, as is customary on HPC systems. At a later stage, this estimate may be refined based on easy to determine parameters, such as input data size and number of resources used, which may be extracted from the logs of previous runs of the workflow. A significant amount of research has been done on estimating application execution time based on limited information. For example, [Smith1998] presents a technique that predicts application runtimes based on historical information of “similar” applications. Search techniques are used to automatically determine the best definition of similarity. In [Gaussier2015], a similar technique is used to fine tune the execution time estimate provided by the user.

3.2 Model Verification

3.2.1 Benchmark Application

An artificial benchmark workflow will be created which allows configuration of the different aspects of a workflow, such as the sizes and locations of in- and output data, pre- or post-processing requirements, the number and type of compute resources required, the execution time of the application, etc. This benchmark workflow can be used to test the functionality or the PROCESS services, determine the initial values of the model, and validate model predictions.

By choosing minimal values for *data transfer* and *execution time* (for example 0 bytes and 0 seconds) the lower bound for the runtime can be determined and the overhead of the PROCESS services can be measured. By submitting large numbers of such workflows, the scalability of the services themselves can be tested. By choosing large values for data transfer an initial estimate of the data transfer capacity between locations can be made.

Similarly, different pre-processing patterns can be tested, ranging from straightforward filtering or conversion to more complex operations such as mixing or transpositions, to create an initial estimate of the Dispel overhead. By varying the target resources of the workflow, an initial estimate of the scheduling delays in different locations can be made.

Once an initial model is available, this benchmark application can be used to validate it by comparing the error rates of the predictions against actual measurements. This will allow us to iteratively refine the model during the course of the project.

3.2.2 Use Case Workflows

As explained above, strong and weak scalability tests may be performed on the use case workflows to determine the limits to their scalability and the initial parameters of the execution time models. Once these parameters are available, an initial execution time model can be created, and its predictions can be verified using the logs of subsequent workflow runs. Consistently measuring the workflow performance and selected key parameters (such as input data size and type and number of resources used) allows the model to be refined further. By default, a simple placeholder model will be used by the PROCESS services. If necessary, a more detailed use case specific model may be created for a use case and provided upon workflow submission.

3.3 Conclusion

In this section we have described the components of a simple predictive model for workflows performance on the PROCESS infrastructure. The main goal of this model will be to verify that the overhead incurred by the PROCESS services (the sum of T1, T2, T4 and T7 in Figure 1) is negligible compared to the cost of data staging (T3 and T8), scheduling (T5) and execution (T6). Using this model, we try to verify if the proposed services are capable of scaling into the exascale range.

4 Measurements

4.1 Platform-wide measurements

In this section, we report the overhead and scheduling measurements on the PROCESS platform in its production prototype.

4.1.1 Overhead measurements

Due to some integration issues preventing us from using certain resources, the overhead measurements are performed for scenarios 1 and 2 and are presented together. The measurements are taken on Prometheus and each value is the average of four consecutive runs whenever possible. Indeed, not all runs lead to data usable for the analysis. The benchmarking uses an event-based approach, where the state transitions allow to extract event durations. Because of the polling required to extract the events, most of the transitions are missed in normal operations, which is good from a production point of view. Unfortunately, this leads to quite a small set of data points for most of our performance analyses. In contrast to D3.2, other overhead factors are measured in addition to T2. The operational conditions of the tests impose frequent polling, whose consequence is a dither of +/-5 seconds in the data, which are plotted in Figure 3 for T2 or submission delay. The main observation is that, except for an outlier at container count 96, the overhead is small, almost constant and independent of the number of containers. Indeed, both the submission and the scheduling delays (see below) are not under the control of IEE; consequently, some jobs get stuck either waiting for or in the queue of the job scheduler on the used HPC systems which leads to occurrences of outliers.

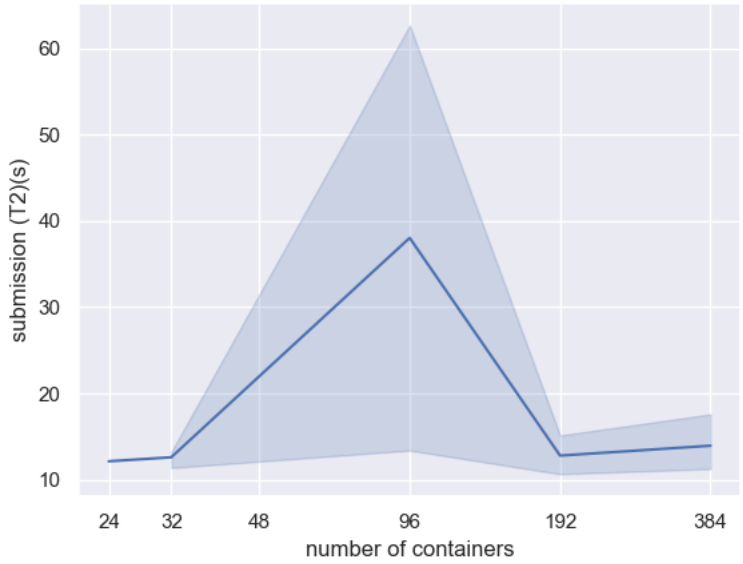

Figure 3: Submission delay (T2) behaviour in PROCESS production prototype

In Figure 4-a (left), the measured values of T2 are grouped by input data size of 10MB, 100MB, 1GB and 10GB which are the test input data sizes. We observe that up to 1GB, the average value of T2 is quite small with little deviation from that value. However, for the 10GB batch, the average value has significantly increased with a large standard deviation. This is due to the outlier at container count 96 shown above. A plot without the outlier is shown in Figure 4-b (right). We can indeed observe that the average submission delays are close within the 2-3 seconds range. The only reason the 10G batch is larger is because of an outlier at container count 96 for this input data size.

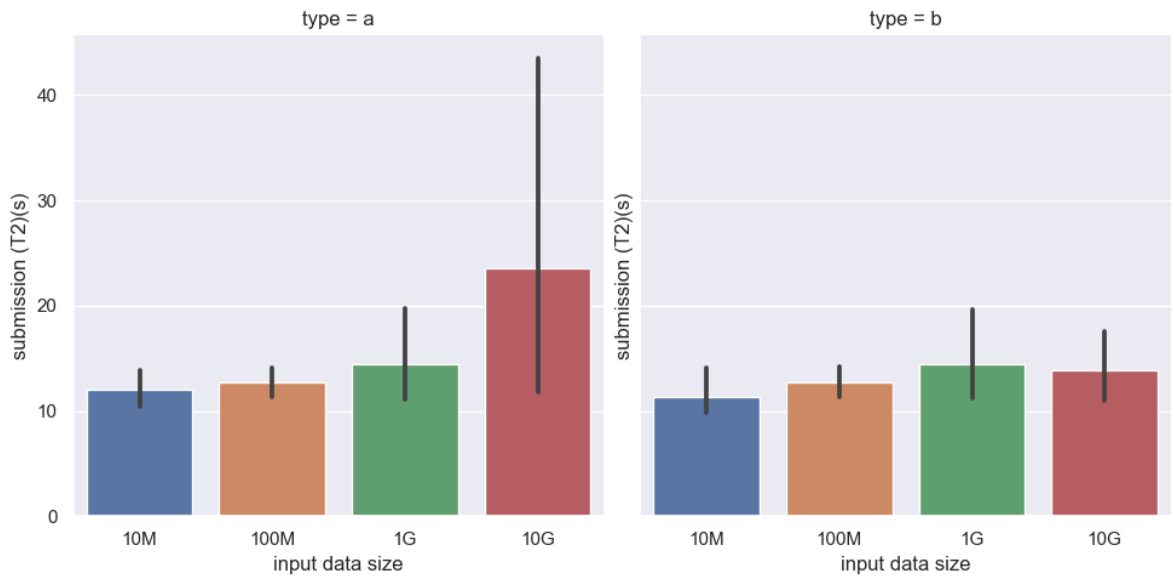

Figure 4: Submission delay in IEE batched by input data size.

We also measure other overhead factors including the initial directory building, which creates a directory structure to hold the input data and the intermediary results, and the implicit staging which involves transferring the output of one step into the input directory of the subsequent step and a clean-up step removing the above directory structure. While the first and last steps may happen on any data processing infrastructure, the implicit staging is specific to IEE; consequently, this is the only one we will consider here. The behaviour of the metric is shown in Figure 5 below. We observe that the implicit staging is of the same order of magnitude as T2 but at a generally lower scale.

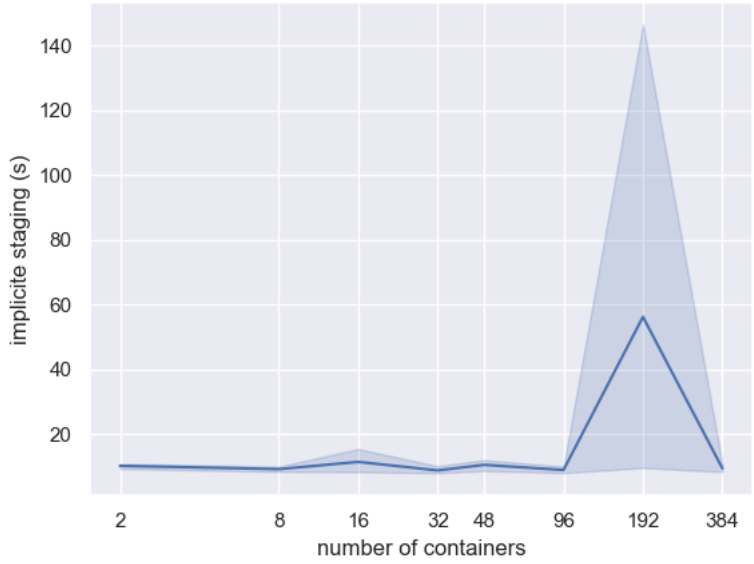

Figure 5: Implicit staging overhead behaviour in PROCESS production prototype.

The cumulative behaviour of T2 and implicit staging is illustrated in the Figure 6 below. The principal observation is that the global overhead is moderate. It culminates at 25 s at container count 384.

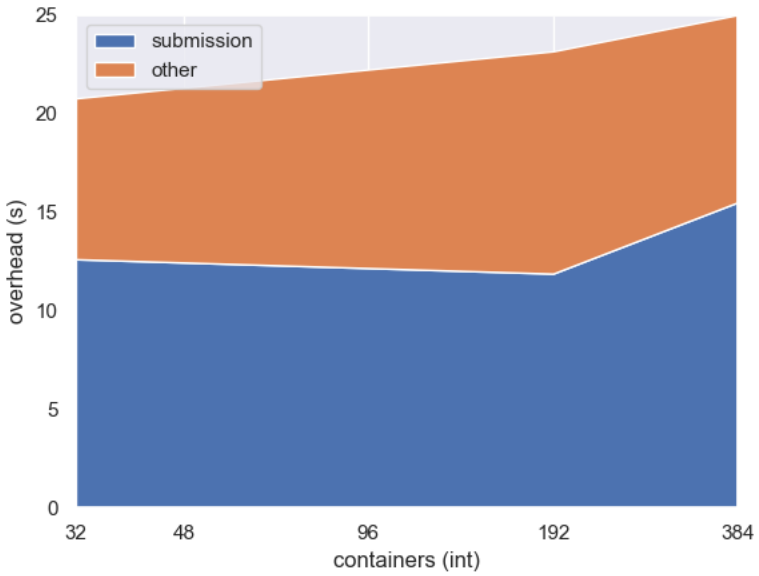

Figure 6: Overall overhead (T2 + implicit staging) behaviour in PROCESS production prototype

4.1.2 Scheduling measurements

Overhead due to scheduling in IEE is measured as queueing times which are plotted in Figure 7 in relation with the number of containers. The measurements are taken in the same conditions as for the overhead and, each measurement is an average of up to four measurements. We observe that scheduling does not harm PROCESS performance as its overhead is the order of tens of seconds for most container counts. A few of the jobs got stuck in the queue, spending there much longer time than average. The job with container count 192 is one of these.

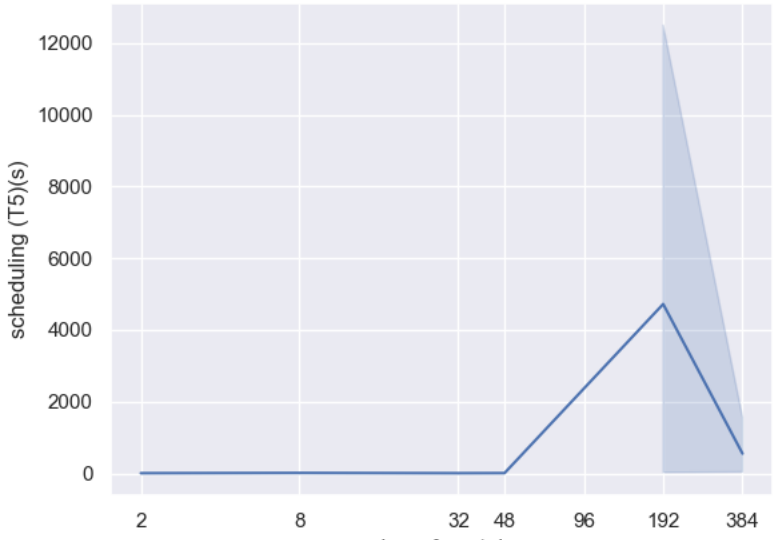

Figure 7: PROCESS scheduling overhead measurements from IEE

A complementary explanation of the scheduling behaviours is given in Figure 8. Just as Figure 7 shows that T5 does not depend on the container count, Figure 8 shows it does not depend on the input data size neither. The delay batches for 1GB and 10GB are both lower than that of 100M, which is where the above outlier happens to be.

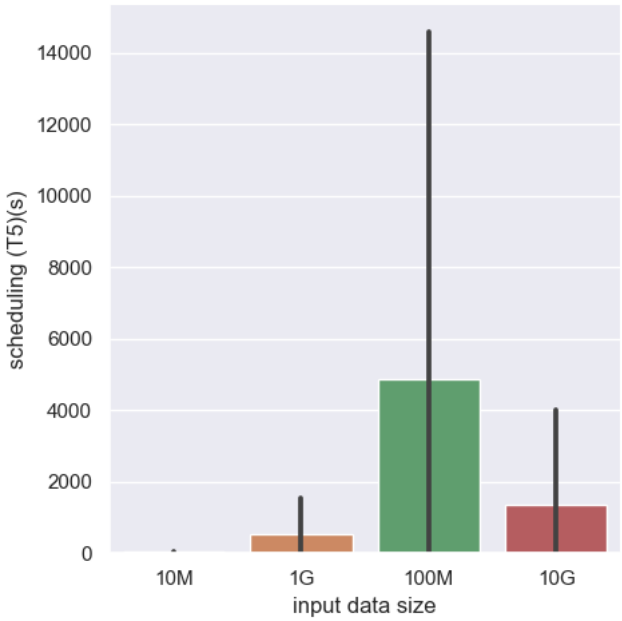

Figure 8: Scheduling delay measurements in PROCESS production prototype batched by input size.

4.1.3 Staging measurements

We also evaluate the staging performance, although this is better done with the data services inside the use cases. However, this gives us a glimpse of their performance in the IEE context. In the latter, the most interesting is the stage-in duration (T3) which involves real data transfer using PROCESS data services, which we report in Figure 9. The stage-out depends on the use case and for the test or validation pipeline it does not involve data services and does not correspond to any useful output. The obvious note is that T3 does not depend on the container count, but rather on the input data size. Indeed, although the transfer durations are almost indistinguishable for up to 1GB, the transfer duration for 10GB clearly increases. This is expected as transfer time only depends on input size and network performance and conditions. The latter is probably responsible for the important spread at container count 192.

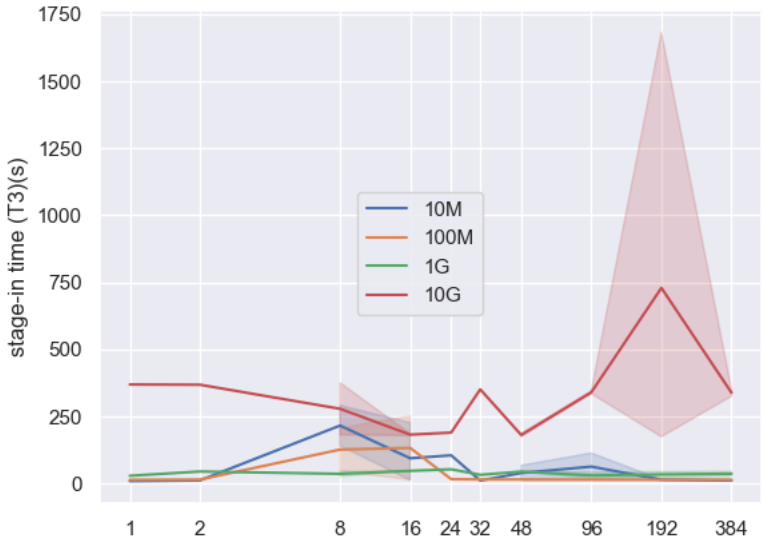

Figure 9: Staging-in measurements in IEE production prototype.

4.2 Use case specific measurements

4.2.1 UC1

Data transfer measurements

Measurements of data transfer rates with the SCP protocol were reported in D8.1. As reported in the deliverable, frequent stalls and broken connections happened frequently in head nodes. The DTN connection from LRZ to AMS and AMS to LISA showed 30% faster transfer than a direct copy (see Table 6 in D8.1, page 11). In Table 3we report the measurements of copying the UC1 Camelyon16 dataset between PROCESS sites including our DTN using gridFTP.

Table 3: Measurements of copying the Camelyon16 dataset between PROCESS sites, with gridFTP protocol in MB/s.

Camelyon 16 30Gb transfer source/destination	AMS-DTN	LISA	LMU-DTN	AGH
AMS-DTN	0	324.17	494.51	25.53
LISA	549.62	0	324.97	0
LMU-DTN	405.32	25.53	0	19.55
AGH	51.07	0	14.71	0

The asymmetry in the measurements is due to different reasons. The lack of open ports of sites, for example, reduced the possibility of having concurrent connections. Moreover, the sites have different upload to download bandwidth ratios. Direct connectivity between Lisa and Prometheus was not possible with gridFTP, since this would require one of the sites to act as a server and have open ports. These many restrictions further emphasize the need for a DTN approach with more performance and programmability for data transfers.

A further analysis concerning the usage of container protocols to program DTNs shows a better amortization of the costs for data transfer (see Figure 10 and Figure 11):

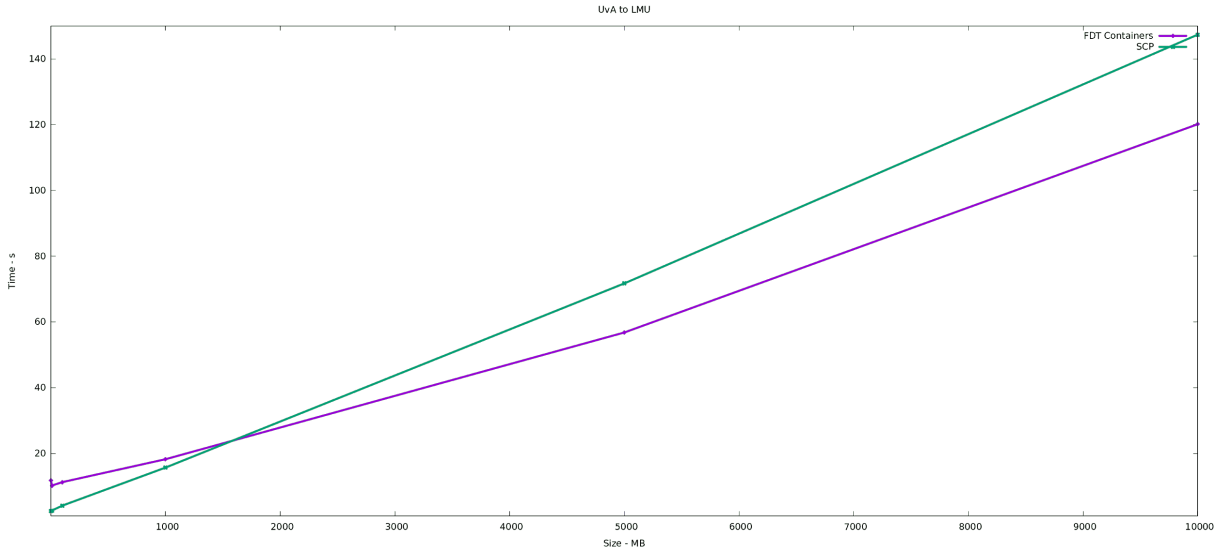

Figure 10: Comparison of the standard SCP protocol and the containerized FDT protocol to transfer from UvA to LMU. Time for transfer is reported against file size in Mb. FDT shows better performance on files larger than 2Gb.

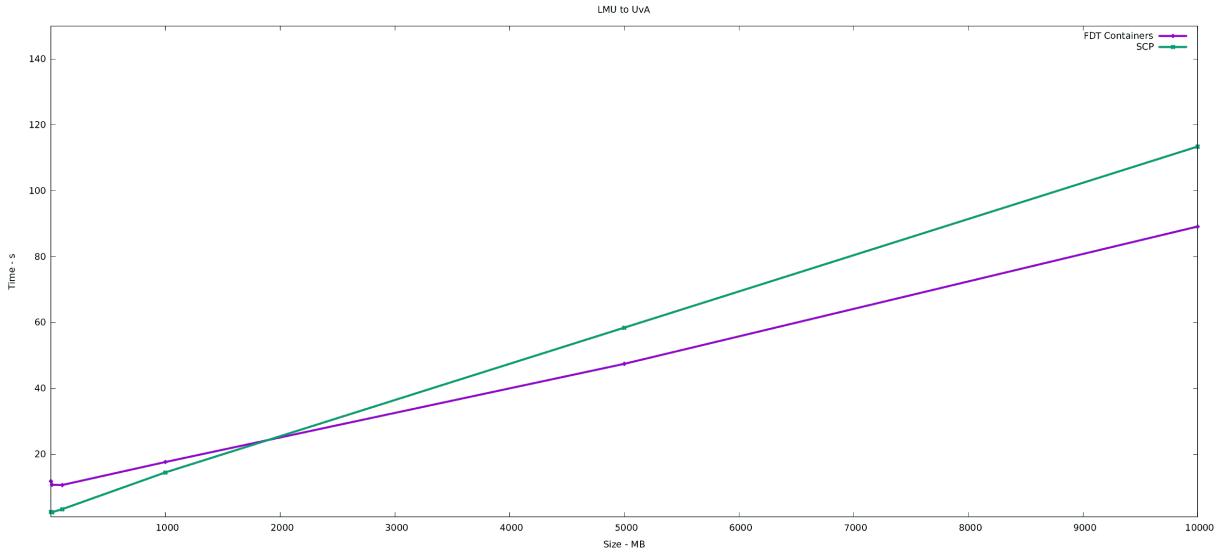

Figure 11: Transfer from LMU to UvA: comparison of the standard SCP protocol and the FDT protocol in a containerized approach. FDT shows better performance on files larger than 2Gb.

Figures show a comparative of using standard SCP protocol to transfer data between DTN nodes vs the dynamically reprogram DTNs with different protocol containers. In this case we deploy FDT protocol inside a container. What is evident is that for files smaller than 2Gb the overhead of deploying the containers on the fly is greater than the transfer time which results in a degradation. For larger files (around 2GB) the overhead is amortized by the better performing FDT protocol which at 10GB data size, FDT is ~22% faster.
Note that this analysis can be applied to the five PROCESS use cases.

Execution time and/or FLOPS measurements vs data size

Initial measurements of the execution time for each software layer were reported in D8.1 Table 2 and 3, pages 8-9. In Table 4 we report the execution time against the size of the data in the data pre-processing step, for each of the two methods proposed, namely random sampling and dense sampling. Measurements were computed on the AGH site in Krakow, Poland. The execution time vs data size is reported in s/Mb or s/Gb to show the scalability to increasingly larger datasets and the gains in computing time which reach up to processing 1 Gb per second.

Table 4: Measurements of execution time vs data sizes for extracting high resolution patches from the Camleyon17 dataset at the PROCESS AGH site.

Method	# patches	WSI coverage	data size	sampling time	execution time (upper-bound)	Execution time vs. data size
Random sampling	500	1 file	144 Mb	0.05 s	49.12 s	0.29 s/Mb
Random sampling (in D8.1, Table 2)	5000	5 files	12 Gb	0.05 s	286.2. s	21.9 s/Gb
Dense sampling	118773	10% of 2% of files	200 Gb	0.004 s	620.31 s	2.8 s/Gb
Dense sampling	1 Mill.	100% of 10% of files	1 Tb	0.004 s	~ 7500 s (2 hours)	1 s/Gb
Dense sampling (sequential at UISAV)	4.5 Mill	100% of 100% of files	7 Tb	0.02 s	~ 739651 s (8.5 days)	106 s/Gb

Table 5: Parallel Model Training lower bounds expressed as number of trained models trained per hour (on 50 Gb of training data)

Number of models	GPUs	Model complexity in number of parameters	Model Training time (upper bound in hours:minutes)
1 x ResNet 50	1 x Nvidia K80	23 Million	07:00
1 x ResNet 50	1 x Nvidia V100	23 Million	06:00
1 x ResNet 101	2 x Nvidia K80	44.5 Million	05:00
1 x Inception V3	1 x Nvidia V100	23.83 Million	07:00
100 x Inception V3	3 x Nvidia V100	7149 Million (7 Billion)	36:00

4.2.2 UC2

Data transfer measurements

Since the benchmarks carried out in D8.1, the services for the data transfer between the LOFAR long-term archive (LTA) and the HPC cluster sites (CYF, LISA, and LRZ) have been further developed. To measure the effect of these developments, part of the benchmarking has been redone. The queuing time and staging time at the side of the LTA has not been benchmarked again, since this mechanism has not been updated and is out of our control. Therefore, the existing results, as outlined in Section 3.2 of D8.1, still remain accurate.

For completeness, we recall here our findings from D8.1. We performed three types of measurements: estimating the queueing and preparation time on LOFAR LTA tape archive, total staging time as a function of total size and data transfer speed from the archive to the HPC sites. We have found the queueing durations quite variable in and across LOFAR LTA locations, those variations being attributed to differing configurations and/or loads at the respective locations. We have also found the staging durations to be variable because of the shared nature of the tape systems. Finally, the data transfers between the LTA locations and the various computing sites are shown to be suboptimal (roughly in the range of 5 to 12 MB/s) and constitute a bottleneck that needs to be overcome. The updated transfer speed benchmark (Figure 12) shows an improved average speed. It is expected that this is due to a switch in the Globus library used for the transfers (from a Java to a C library) and the use of concurrent transfers, instead of transferring files one by one.

In most cases, the transfer speed is increased to roughly the 80 to 120 MB/s range. With an exception being the transfers between LISA and the Amsterdam LTA location, which is significantly higher, but not changed since the previous benchmark. This is expected, since both locations operate on SURFsara's optimized grid infrastructure[11]. The other exception are the transfers from the Ponzán LTA location to the LRZ and LISA clusters, although a modest speed improvement has been achieved. It remains unclear why this is not at the same level as the other transfers. It may be due to the particular (public) network in place or load thereof, configuration, temporary disruption.

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

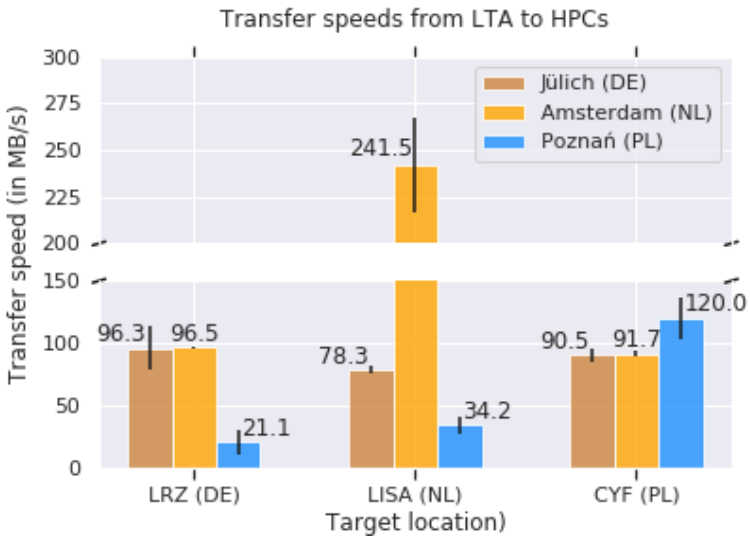

Figure 12: UC2 data staging and transfer measurements.

Although we were able to improve the transfer speeds, the conclusion as provided in D8.1 still holds. In the extreme-scale data services that we are targeting, the current speeds are still too low compared to the size of the files that we would like to transfer. Therefore, the transfer speed is still considered a bottleneck, when not using an optimized network, e.g. with DTNs.

Execution time and/or FLOPS measurements vs data size

For UC2, we only measure the execution time for the time being as the most intensive components capable of generating high FLOPS values are currently sequential or multithreaded. The wall clock times of the main steps of the data reduction pipeline are given in Table 6. The main conclusion to draw from the very high magnitude of these values is that the overhead due to scheduling and interaction between platform components as seen in the previous section is negligible.

Table 6: Wall clock times of the main steps of the data reduction pipeline.

step	data size (GB)	execution time (s)
calibrator DI	25	8534
target DI	433	11909
init-subtract	76	37212
DD2 (FACTOR)	76	~5d

4.2.3 UC4

Data transfer measurements

Since there are still ongoing negotiations with our customer about the usage of their database, the measurement of real data transfer cannot happen during the project. To be able to run the model training container we prepared a test data generator as already mentioned earlier in deliverable documents. The test data generator is able to generate the data directly into the PROCESS environment running at UISAV. We have taken the measurements of how long the data generation takes which also measures the times of inserting data into HDFS. Please see Table 7 and Figure 13.

The average times for inserting batches of 1 million records were oscillating near 50ms. There is a noticeable increase in the insertion time during a few first inserts which application does where many inserts take more than 100ms and some even take about a second. There are also occasional spikes of about 500ms inserts. But with a large number of inserts done any of the mentioned spikes are negligible.

Table 7: UC4 data generation time.

Records amount	Generation time [min]	Records generated / s
10,000,000	0.33	499,226.20
20,000,000	0.85	391,910.96
50,000,000	1.79	465,783.54
100,000,000	2.45	679,564.81
200,000,000	4.66	715,241.07
250,000,000	5.85	712,147.24
300,000,000	6.93	721,780.20
350,000,000	8.09	721,192.98
400,000,000	9.38	710,389.99
450,000,000	9.77	767,523.85
500,000,000	11.25	740,792.32
1,000,000,000	22.40	744,071.42
2,000,000,000	45.07	739,653.72

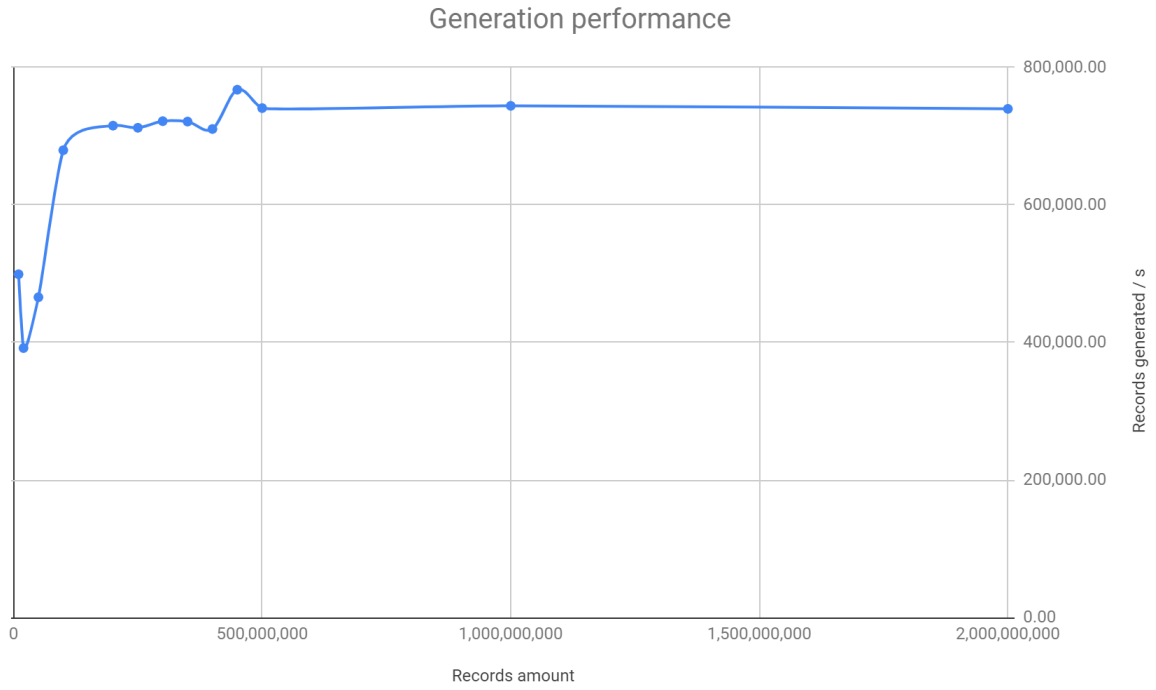

Figure 13: UC4 data transfer measurements.

Execution time and/or FLOPS measurements vs data size

Based on the generated data we measured the execution time of training the machine learning models. Two types of models were generated, a random forest model and a deep neural network model. Both models were trained with the same dataset. Measured time was increasing exponentially for random forest model and linearly for deep neural network model. The biggest issue was that both the application and the H2O cluster [H2Ocluster] were deployed in the same container. Furthermore, the H2O cluster sometimes did not have enough memory to load all the data. The maximum amount that we were able to load were bookings and services generated for 350 million flights (Table 8 and Figure 14, Figure 15).

Table 8: UC4 model generation time.

Records amount	Model training time [H:MM:SS]	
	Random Forest Total	Deep Neural Network Total
10,000,000	0:00:05	0:02:04
20,000,000	0:00:06	0:05:19
50,000,000	0:00:07	0:13:55
100,000,000	0:00:09	0:30:38
200,000,000	0:00:12	1:02:24
250,000,000	0:00:19	1:14:24
300,000,000	0:00:24	1:56:24
350,000,000	0:00:40	2:16:48

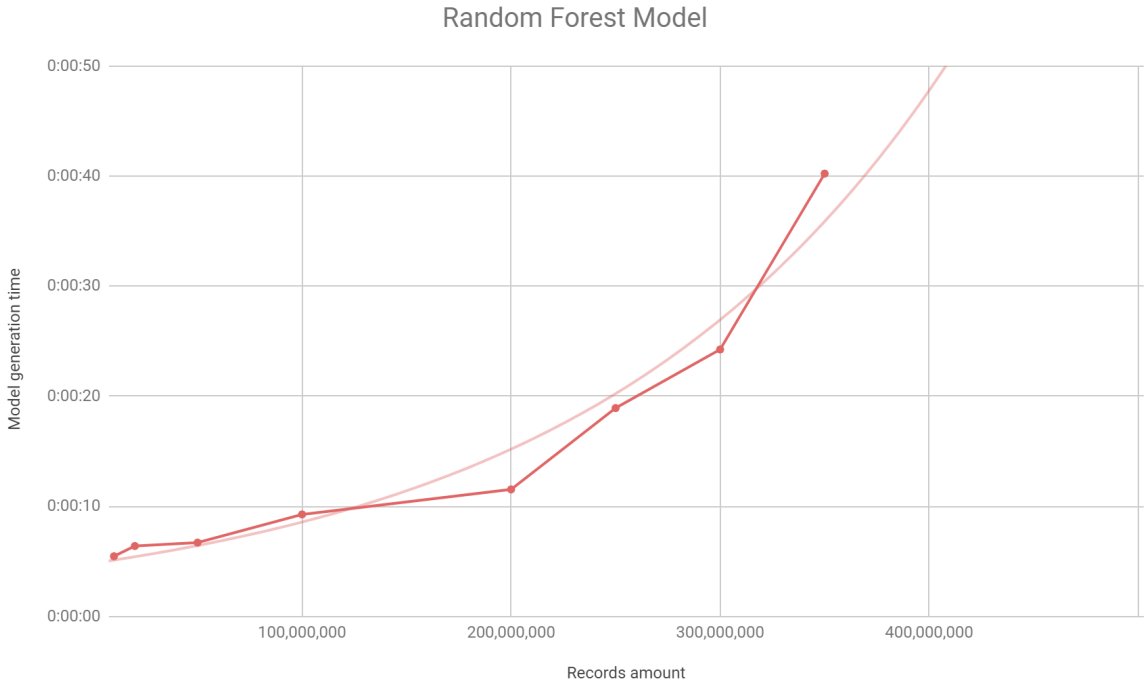

Figure 14: UC4 Random Forest model training time over sample size.

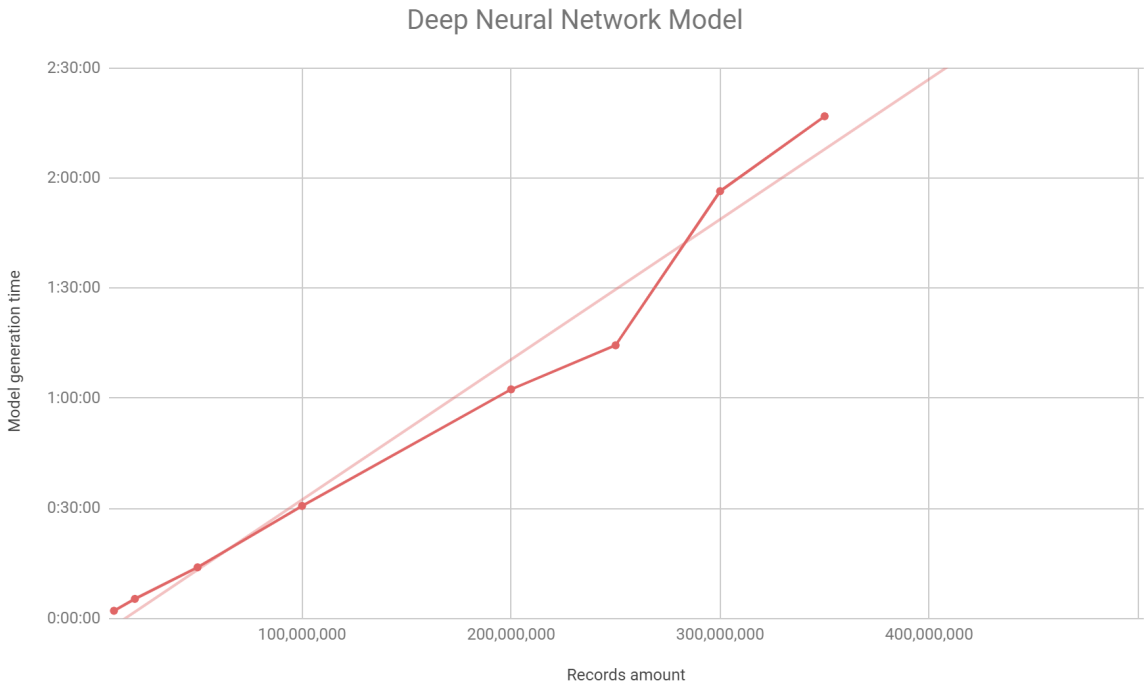

Figure 15: UC4 Deep Neural Network model training time over sample size.

The version of Use Case 4 written in R language had problems with very long build times which hindered the possibility of fast prototyping. Because the H2O framework library is also available for Python it was decided that the whole application could be rewritten in that language. Performance of both build times (Table 9) and model training times were measured (Figure 16, Figure 17) to confirm that build times are now faster and model training times are similar to the R version of the application. It was measured by using an external H2O cluster with a minimal set of 2 nodes, 4 cores and 2GB of

memory in total. Build times were significantly better but there was a slight decrease in performance of calculating deep neural network models.

Average build time of Docker container [H:MM:SS]: ()

- without cache (**R** version) - **0:36:10**
- without cache (**Python** version) - **0:25:40**
- with cache, code change, no Dockerfile change (**R** version) - **0:16:18**
- with cache, code change, no Dockerfile change (**Python** version) - **0:02:36**

Table 9: UC4 Python and R versions comparison.

Records amount	DRF (Python)	DRF (R)	DNN (Python)	DNN (R)
20,000,000	0:00:07	0:00:08	0:03:33	0:02:58
50,000,000	0:00:10	0:00:11	0:08:35	0:07:24
100,000,000	0:00:12	0:00:12	0:16:42	0:14:58
200,000,000	0:00:20	0:00:20	0:30:49	0:24:55
350,000,000	0:00:25	0:00:26	0:49:38	0:44:56
400,000,000	0:00:29	0:00:33	0:52:28	0:46:20
450,000,000	0:00:38	0:00:32	1:04:02	1:02:24
500,000,000	0:00:51	0:00:35	1:15:46	1:01:12

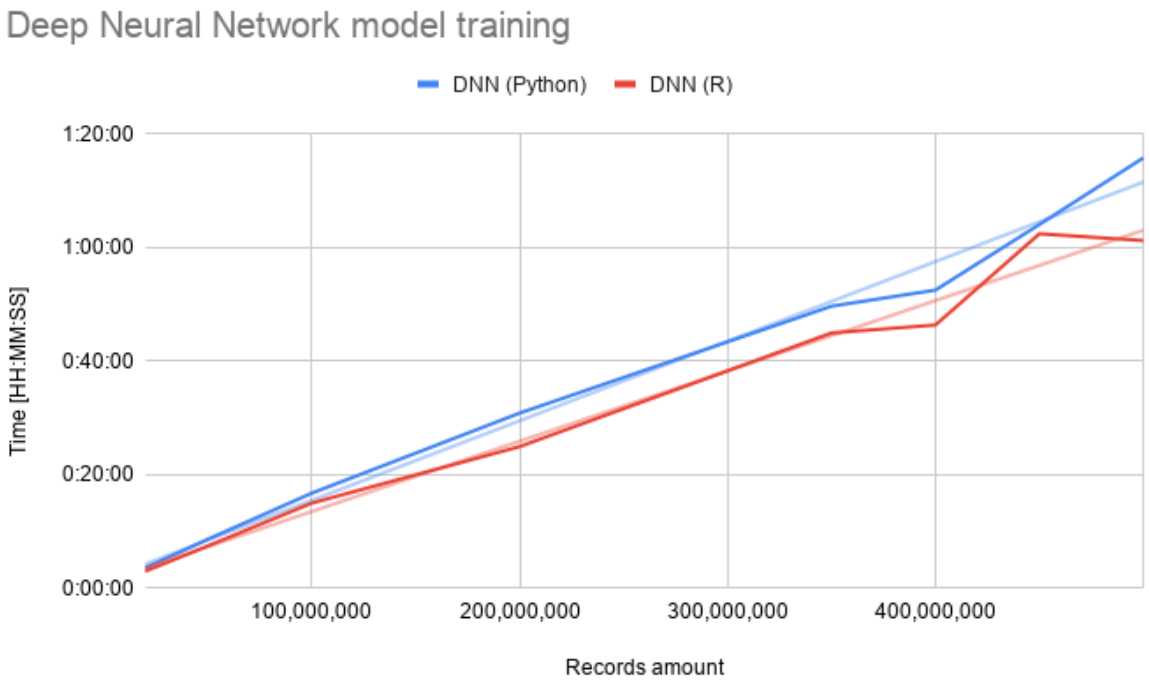

Figure 16: UC4 Python and R versions Deep Neural Network model training comparison.

Figure 17: UC4 Python and R versions Random Forest model training comparison.

4.2.4 UC5

Data analysis based on satellite data from the Copernicus project to model agricultural usage is an important task and impacts not only academic but also industrial research. Within PROCESS this Use Case is driven by the proprietary SME software, PROMET, which is executed as a closed source application on a distinct target system. As reported in D8.1, Section 6, this execution is connected to PROCESS via an API as an intermediary between the IEE and the target system. The resulting workflow overview was presented in D8.1, Figure 9, and detailed in a sequence diagram shown in D8.1, Figure 10.
This sequence diagram allows a very precise mapping of the described measurements T1 to T8 as explained in the following Table 10.

Table 10: Use Case 5 measurements.

TX	Name	Realisation
T1	Configuration	Each run is configured manually in the IEE, where different values of the software can be set. Additionally, a demonstration workflow can be executed without any interaction.
T2	Deployment Strategy	The selection of storage and computing sites is not applicable to this use case, since the software is bound to a closed source target system. The deployment is done with one API call.
T3	Stage-In	Input data is not transferred to the target site, since it is proprietary and not publicly available.
T4	Container selection	Within the deployment API call, also a specific version of the software can be chosen within T1.

T5	Schedule	Although scheduling and execution is part of the closed target system, these measurements are provided by the use case.
T6	Execution time	
T7	Stage-Out Strategy	The output is accessible by the end user through the IEE. The data storage is predefined and the transport is done via LOBCDER. The measurement therefore includes only the transfer time.
T8	Stage-Out	

Based on the presented measurements the resulting data is shown in Figure 18. The plot includes error bars with the standard deviation of each measurement, which is quite high in T1, T5 and T6. T1 is dependent on the user's input and configuration and differs therefore based on several reasons, e.g. amount of choices, experience, distraction. The scheduling in T5 of the actual job is dependent on the load of the whole system and is not controllable by PROCESS. Although the execution time of the software is slightly influenced by the overall load on a HPC cluster, the displayed spread is based on the execution of very different configurations and therefore resulting in very different execution times. The interaction of the API and IEE is negligible, since it requires only one call to start the deployment and another one for confirmation. T3, T4 and T7 are not applicable to this Use Case. T8 measures the transport of a 10.4 GB output file through LOBCDER including the necessary protocol steps. The average bandwidth is 0.34 GB/s which is limited by the transport network, which is not controlled by PROCESS.
Overall, the UC5's measurements show a very efficient usage of the PROCESS ecosystem. Throughout this deployment no overhead introduced by PROCESS could be observed. All queuing times and transport delays are directly connected to systems and networks not controlled by PROCESS.

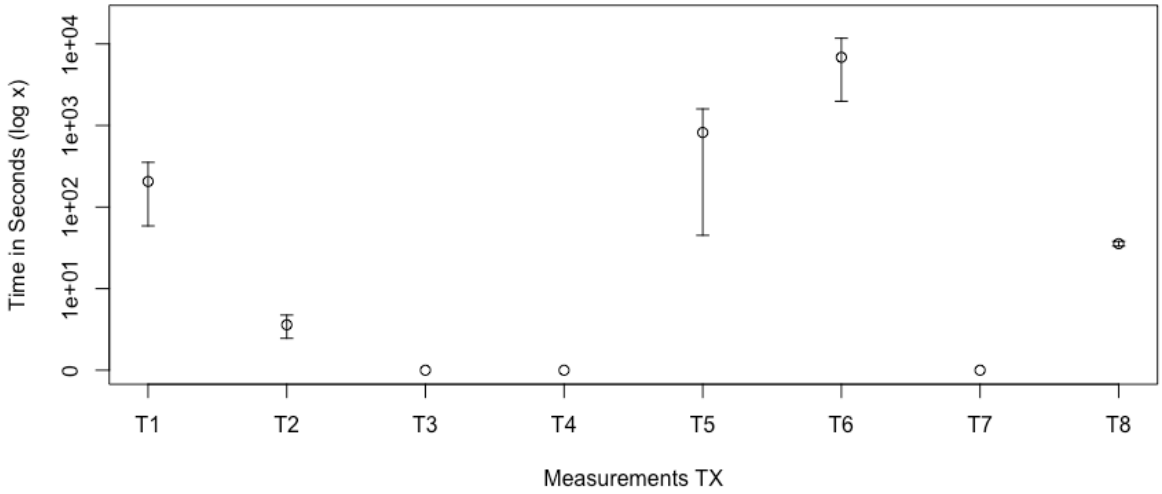

Figure 18: UC5 Measurements.

Comparing the actual scheduling (T5), computation (T6) and data transfer (T8) time on the compute resource with the overall deployment and execution time logged in the IEE, we can reduce the introduced overhead to seconds. This overhead is limited to the communication between the IEE and the UC's API.

5 Application of the Prediction Model to actual Measurement Results and Conclusion

5.1 Overhead model and projection

Using the measurement data collected in Section 4, we model the behaviour of the platform overhead using regression analysis to get insight into the data. However, because of the occurrence of outliers here and there, we do use robust regression to down weight extreme values. Modelling results for overhead are shown in Figure 17. The linear model (equation: *overhead (o) = 0.011 * (number of containers (c)) + 20*) of the data shows a very slow variation of the overhead in function of the number of containers, which is very important for PROCESS scalability. Although the very small value of the slope implies that the increase is very slow, so we can consider this overhead as practically constant and independent of the number of containers. To put this in context, we can confidently assert that the overhead of processing the entire LOFAR LTA archive (around 1800 observations of 16TB) would only be about 40 seconds.

Figure 19-a (left) plots the residual values for each observation and allows to check whether the regression model is appropriate for the dataset. If it does, then the values should be randomly scattered around the value y = 0. As this is what we observe in Figure 19-b (right), we are confident that our approach is appropriate.

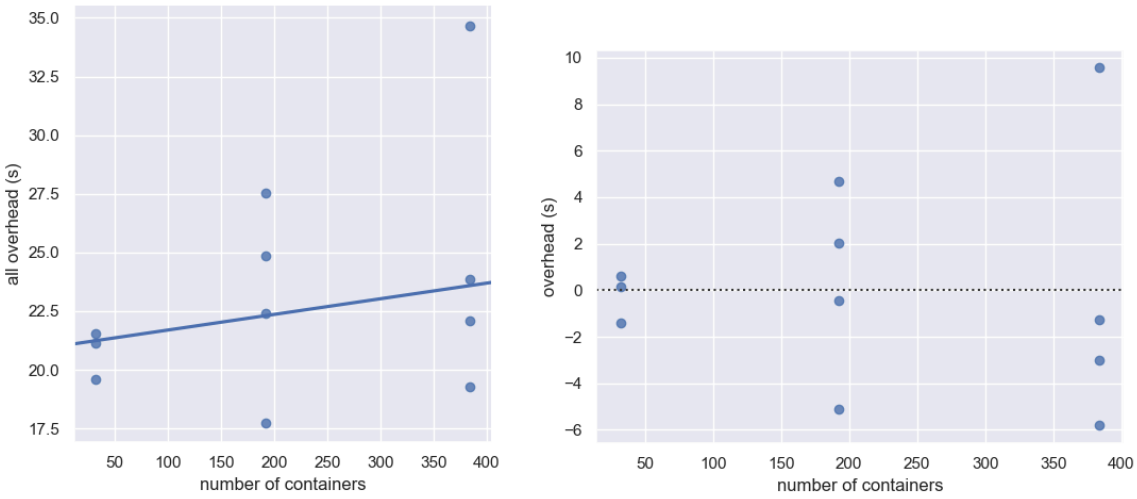

Figure 19: PROCESS T2 overhead models.

5.2 Scheduling model and projection

Similar to the general overhead, we use the measurements in Section 4 to model the behaviour of the platform scheduling overhead. As illustrated in Figure 20, the IEE model shows a moderate variation of the scheduling overhead proportionally to the number of containers (*o = 1.67* c + 160*) . The principal observation is that the scheduling due to PROCESS creates some burden, the latter is moderate and is not under the control of PROCESS services. Indeed, depending on the resources and load on the used HPC clusters, some jobs get stuck in the workload management system queues for unpredictable durations. And the more jobs, the more probable some will get stuck.

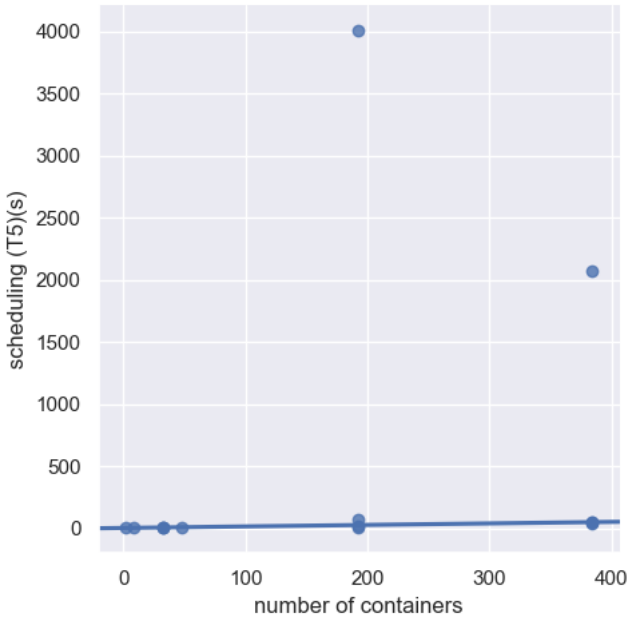

Figure 20: PROCESS scheduling overhead models in IEE production prototype.

5.3 Data transfer model and projection

In section 4.1.3, we showed that the staging performance is independent of the container count. This is illustrated by the linear model of the staging delays in function of the number of containers as a practically horizontal line in Figure 21.

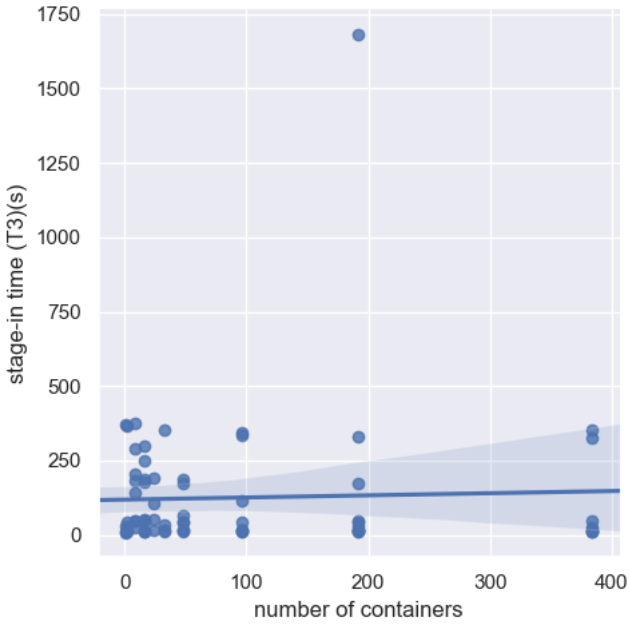

Figure 21: PROCESS staging-in delay model in IEE.

We know that the staging (in and out) performance depends instead on the input data size whose linear model is shown below in Figure 22. The equation of the shown linear model is *T3 = 0.032 * M + 39* where *M* is the size of the input data in MB.

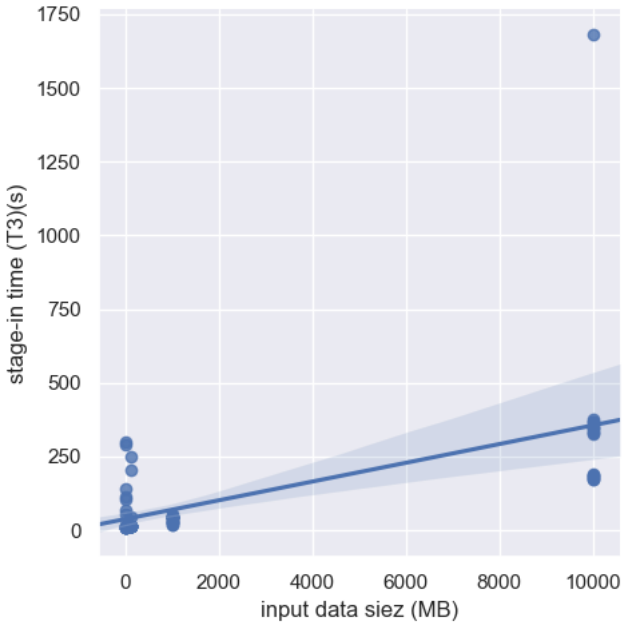

Figure 22: PROCESS staging-in delay model in IEE.

According to the linear model, it would take on average about 359s to transfer 10GB of data which makes for an average speed of about 27.85MB/s; at this speed, it would take 574,506,283 seconds (or 18 years 79 days 9 hours 4 minutes and 43 seconds!) to stage in a full LOFAR observation of 16TB.

5.4 Conclusion and discussion

In this section we model the measurements detailed in section 4. Mainly, three models have been built, for PROCESS platform overhead, scheduling overhead and data staging and transfer.

The platform overhead model validates the choices made in PROCESS architecture and implementation by exhibiting quite a low burden even in case of high load in terms of number of concurrently running containers.

The scheduling overhead model also shows that the scheduling due to PROCESS creates a moderate burden and does not constitute a bottleneck. Finally, the staging model shows that staging can take quite some time, especially for UC2. Unfortunately, it is beyond the control of PROCESS. The very low transfer rates observed for the staging-in is likely due to throttling of local data transfer as the staging uses a shared file system.

6 Conclusion

In this deliverable and the previous related documents we laid down the foundations for the definition and use of a predictive model based on clearly defined performance indicators and scenarios. We briefly reviewed relevant approaches to performance modelling and devised an approach for PROCESS based on a combination of measurements, micro benchmarking and analytical modelling.

An analysis of the architectural components clearly identified the performance indicators or measurands and the scenarios in which they are measured. Then, the measurands were categorized into overhead, attributable to PROCESS, and otherwise, including scheduling and staging. The main goal of the predictive model was to verify that the overhead incurred by the PROCESS services is negligible compared to the other cost factors and that these services are capable of scaling to the exascale range.

The performance indicators were measured across different PROCESS service prototypes and use cases, culminating at the production prototype in this deliverable. Each time, the three main categories (overhead, scheduling and staging) of measurands are modelled and projections to the exascale realised. Our results show that the overhead of the PROCESS platform is generally found to be low, validating the architectural choices made for the project. The scheduling overhead is generally shown to be moderate, but out of the control of the PROCESS project. Finally, the last metric consistently shows that data staging is a bottleneck, especially for use cases involving transfer of large datasets such as UC2. This data transfer performance is highly dependent upon external components such as interconnection networks which are out of scope of PROCESS. Solutions for optimising network performance such as data transfer nodes and FTS are being investigated and implemented.

7 References

[PMO] Performance Modelling, David Henty, EPCC, The University of Edinburgh [online: http://www.archer.ac.uk/training/course-material/2018/07/ScaleMPI-MK/Slides/Performance Modelling.pdf]

[Liu2017] Liu, Z., Balaprakash, P., Kettimuthu, R. and Foster, I., 2017, June. Explaining wide area data transfer performance. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing* (pp. 167-178). ACM.

[Nurmi2007] Nurmi, D., Brevik, J. and Wolski, R., 2007, June. QBETS: queue bounds estimation from time series. In *Workshop on Job Scheduling Strategies for Parallel Processing* (pp. 76-101). Springer, Berlin, Heidelberg.

[Smith1998] Smith, W., Foster, I. and Taylor, V., 1998, March. Predicting application run times using historical information. In *Workshop on Job Scheduling Strategies for Parallel Processing* (pp. 122-142). Springer, Berlin, Heidelberg.

[Gaussier2015] Gaussier, E., Glesser, D., Reis, V. and Trystram, D., 2015, November. Improving backfilling by using machine learning to predict running times. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (p. 64). ACM.

[H2Ocluster]
https://h2o-release.s3.amazonaws.com/h2o/rel-lambert/5/docs-website/deployment/multinode.html