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

D3.2	Performance modelling and prediction		
Project:	PROCESS H2020 – 777533	**Start / Duration:**	01 November 2017 36 Months
Dissemination[1]:	PU	**Nature[2]:**	**R**
Due Date:	31 October 2019	**Work Package:**	**WP 3**
Filename[3]	PROCESS_D3.2_PerformanceModellingAndPrediction-ProgressReport_v1.0.docx		

ABSTRACT

This deliverable is an update of D3.1 and based on its initial content. It refines the performance modelling and prediction approaches outlines in D3.1 based on the results obtained in the project so far and will be completed in D3.3, the next follow up deliverable that will be based on D3.2.
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]	Graziani, Mara; Müller, Henning	HES-SO	Coordinator
Contributing Authors[5]	Meizner, Jan; Nowakowski, Piotr	AGH	Writer
	Höb, Maximilian; Schmidt, Jan	LMU	Writers
	Madougou, Souley; Maassen, Jason	NLESC	Writer
	Valkering, Onno; Belloum, Adam	UvA	Writer
Reviewer(s)[6]	Jungblut, Pascal	LMU	Reviewer
	Maassen, Jason	NLESC	Reviewer
Final review and approval	Höb, Maximilian	LMU	Coordinator

Document History

Release	Date	Reasons for Change	Status[7]	Distribution
0.0	31.01.2019	Final Version of D3.1	Draft	
0.1	06.09.2019	Chapter definition	Draft	
0.2	16.10.2019	Initial Measurements	Draft	
0.4	28.10.2019	Review-Process	In Review	
1.0	31.10.2019	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 D3.2 updates the approach to model the performance of the PROCESS infrastructure and its possible scalability towards exascale workflows. Based on D3.1 this deliverable D3.2 and will prepare the final D3.3 and 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 sub-sets of a given data set on different local nodes and in 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 need 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 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
A very simple sample application running through the complete PROCESS architecture and gathering first results will not contribute to the fundamental performance model. Nonetheless, microbenchmarking will be used to identify performance bottlenecks in the PROCESS architecture and assist in debugging and verifying its correctness.

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 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 datacentres, 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 datacentres.

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:

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.

[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

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.

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.

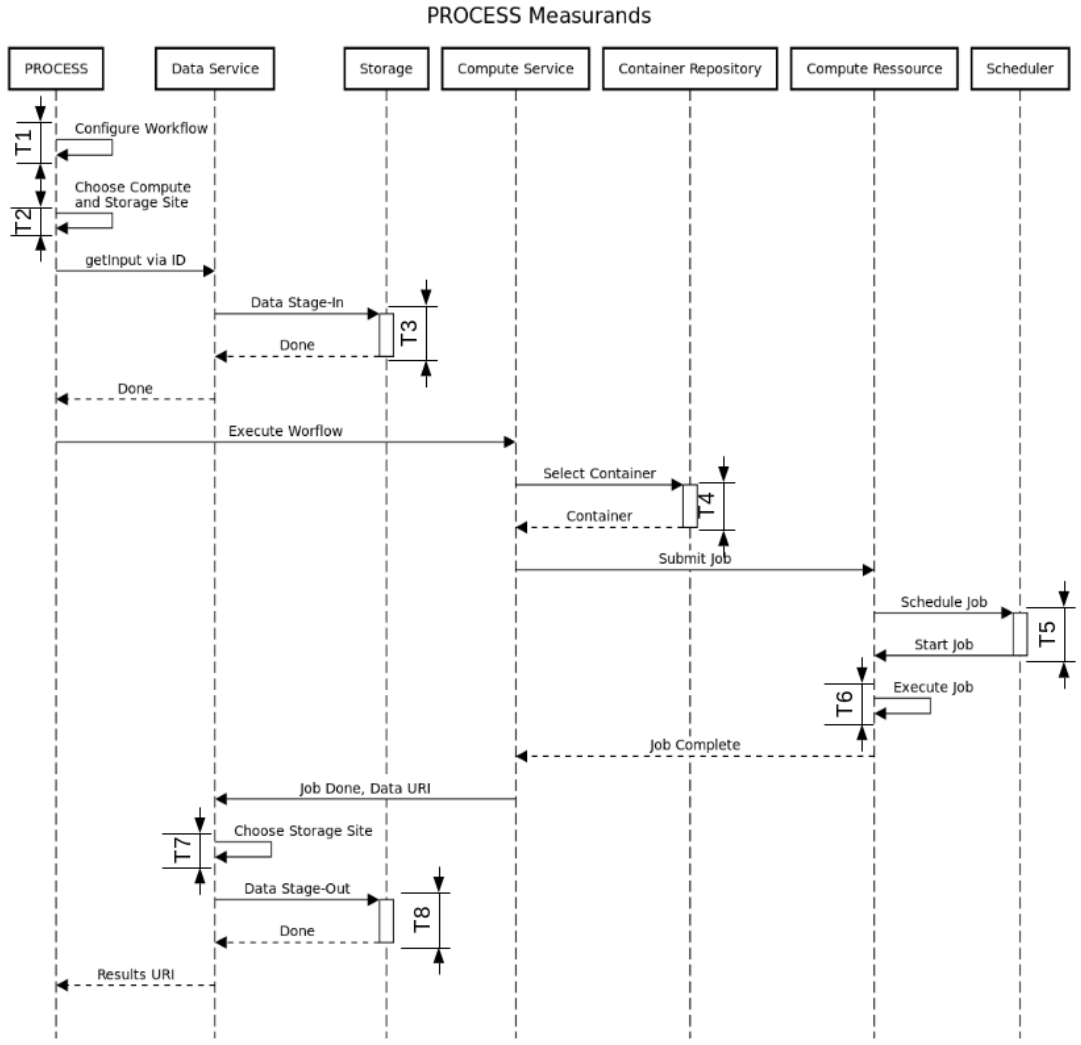

Figure 1: Sequence diagram describing the steps involved in execution of a typical application scenario

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.

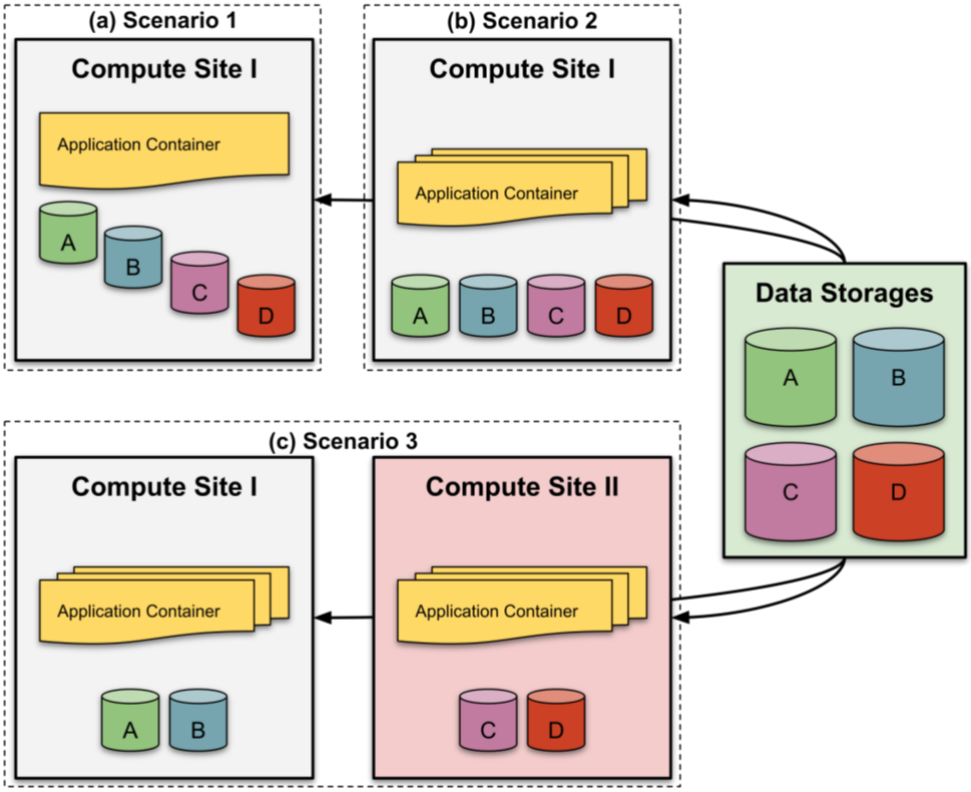

Figure 2: Three measurement scenarios

***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:

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

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 dependent on the scale of the compute resource which are targeted by the 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 source, 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] present 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 its 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) allow 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 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 on 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 current implementation. We also report measurements within UC2 environment which includes Xenon[11] and the common workflow language (CWL)[12] for job submission.

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 three consecutive runs. For reasons detailed in D4.3, only T2 is measured and reported. The operational conditions of the tests impose frequent polling whose consequence is a dither of +/-2.5 seconds in the data, which are plotted in Figure 3. The main observation is that, except the erratic behaviour caused by the dithering at lower container count, the overhead is small, almost constant and independent of the number of containers.

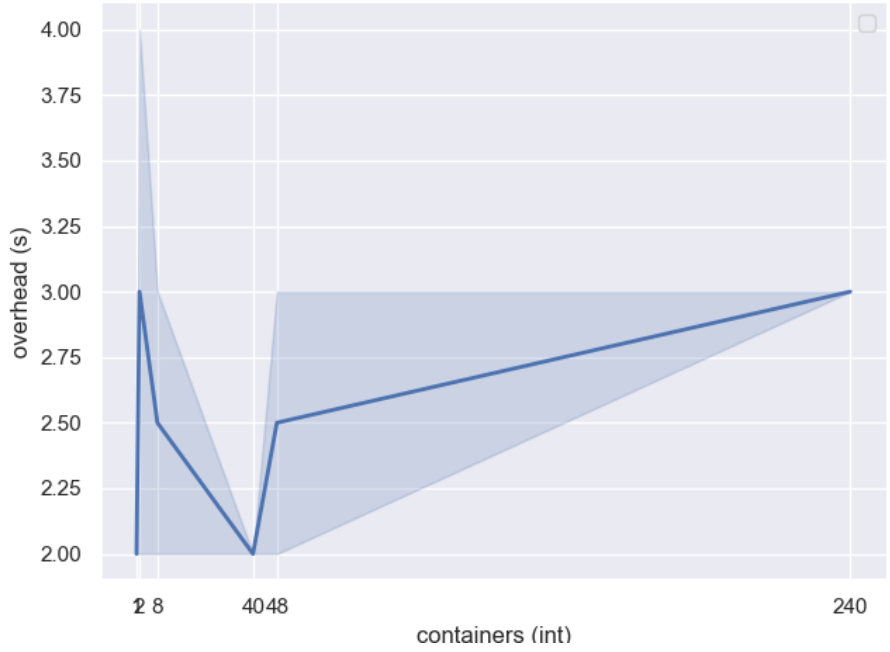

Figure 3: PROCESS platform overhead measurements from IEE. Currently, the only measured is the pipeline submission delay which corresponds to T2.

In UC2 environment, we identified a few factors contributing to overhead. As described in D2.2, current implementation uses a Web portal to select the dataset and to launch the processing. The latter needs to be specified as a CWL workflow whose individual steps are run on given computing site (s) using Xenon and Xenon-flow, a CWL wrapper. Both these tools and the Web portal introduce overhead described below.

Every container is meant to run a reduction pipeline for one observation, but because we are focusing on the overhead, just as in deliverables D4.3 and D8.1, we are not interested in the

[11] https://github.com/xenon-middleware/xenon.git
[12] https://www.commonwl.org

actual computations. Consequently, each step in the pipeline involving computations is replaced with an execution of the validation container developed for validation in D4.3. For Scenario 2, we run consecutively 1, 2, 4, 8, 16 and 32 containers on one computing site. In Scenario 3, we plan to use as much per participating site. All measurements are repeated three times and then averaged. The results are summarised in Figure 4. *Frontend* and *stage-in* overhead is specific in using the Web portal and Xenon-flow and constitute T2. As of the *stage-out* overhead, it corresponds to T7 and is also specific to Xenon-flow. We observe that the individual type of overhead is quite small, but their aggregated value appears almost linear with the number of containers. This linearity is confirmed and eventually detailed in the modelling part in Section 5.

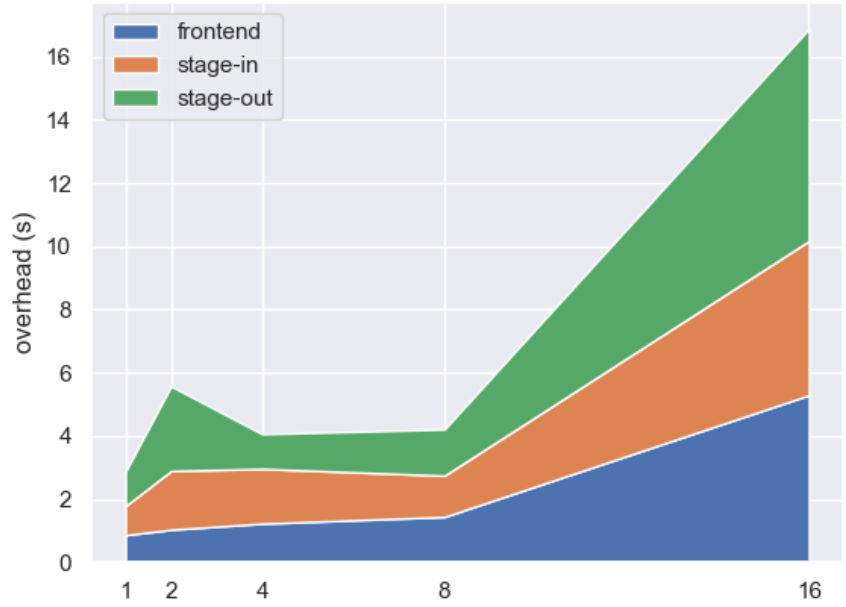

Figure 4: Overhead in current UC2 implementation using Xenon-flow for job submission. Three types of overhead are identified and shown individually, then summed in overall overhead.

4.1.2 Scheduling measurements

Overhead due to scheduling in IEE is measured as queueing times which are plotted in Figure 5 in relation with the number of containers. We observe that scheduling does not harm PROCESS performance as its overhead is the order of seconds to tens of seconds up to 240 containers.

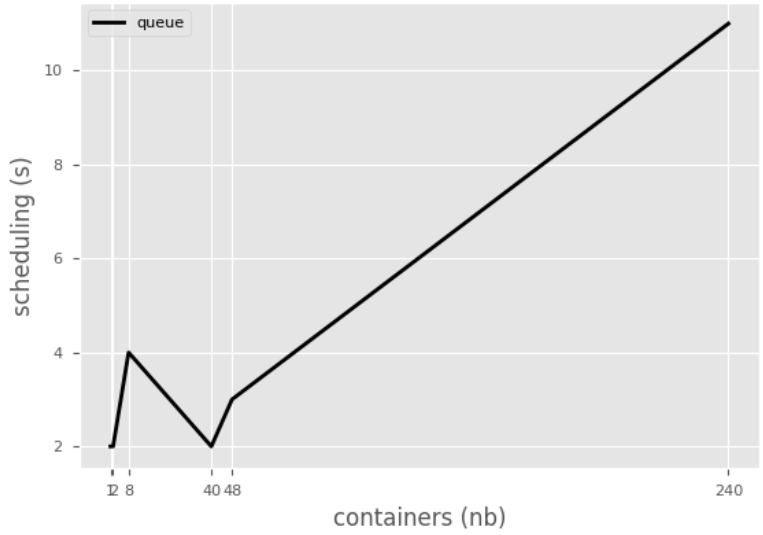

Figure 5: PROCESS scheduling overhead measurements from IEE. It consists of various queueing delay corresponding to T5

We also measure the scheduling overhead in UC2 environment. This overhead is induced by the interaction between Xenon-flow and the HPC system workload management system (WMS) via Xenon. While most of it can actually be accounted for the WMS, some of it is undoubtedly due to the interaction. The variation of this overhead in function of the number of containers is shown in Figure 6. We can see that it has the same profile as the overall overhead above, but in a lesser extent.

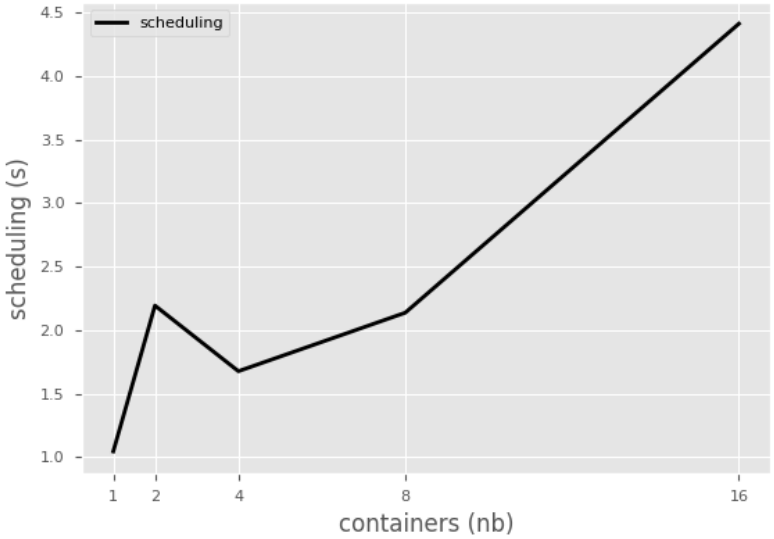

Figure 6: Overhead in UC2 due to interaction with the HPC workload management system to which some of this is accounted for.

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 3 we report the measurements of copying the UC1 Camelyon16 dataset between PROCESS sites including our DTN using gridFTP.

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

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

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. 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.

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 large datasets and the gains in computing time which arrive up to processing 1 Gb per second.

Method	# patches	WSI coverage	data size	sampling time	average execution time	Execution time vs. data size
Random sampling	500	1 file	144 Mb	0.05 s	41.75 s +-7.37 s	0.29 s/Mb
Random sampling (in D8.1, Table 2)	5000	5 files	12 Gb	0.05 s	263.3 s +-22.9 s	21.9 s/Gb
Dense sampling	118773	10% of 1 file	200 Gb	0.004 s	573.31 s +-47.0 s	2.8 s/Gb
Dense sampling	6 Mill.	100% of 5 files	1 Tb	0.004 s	~ 7500 s	1 s/Gb

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

4.2.2 UC2

Data transfer measurements

The benchmarks carried out in Section 3.2 of D8.1 are still valid since none of the involved data service have seen any new developments. Especially, the much-awaited data transfer nodes (DTN) that can heavily impact the transfer performance were not yet available for this use case.

For completeness, we recall here our findings from D8.1. We performed three types of measurement: 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 and cannot be controlled. Finally, the data transfer rates between the LTA various locations and the various computing sites are shown to be suboptimal and constitute a bottleneck that needs to be overcome.

To conclude this section, we report on data transfer measurements using modest datasets performed while benchmarking the platform overhead. The measurements are relative to staging-in input data and staging-out of results for two small datasets of size 10MB and 1GB, respectively, while varying the number of containers running on the platform. The results are summarized in Figure 7. We know the transfer time depends on the data size and is a challenge for PROCESS, the principal observation convoyed by this figure is that time is not dependent on the number of containers. Finally, as in D4.3, stage-in takes longer than stage-out as the former includes transfer time from wherever is the input data is located to Cyfronet in opposition to the latter which does not.

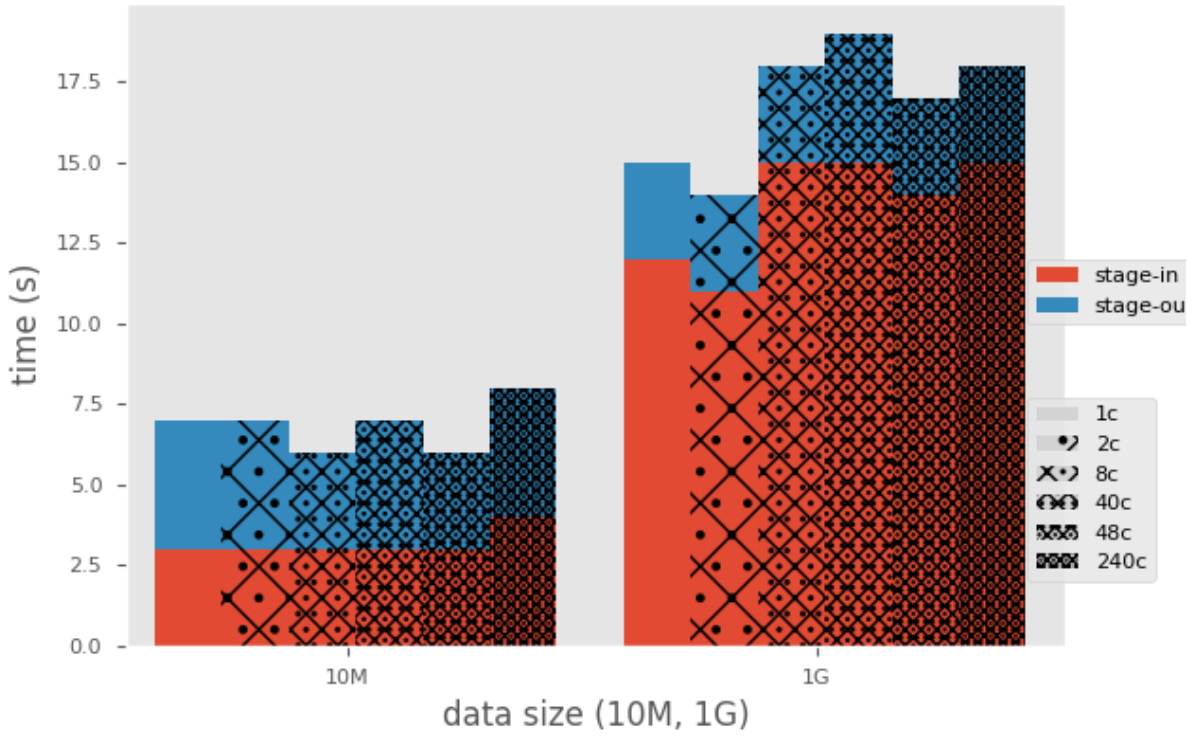

Figure 7: PROCESS general data transfer performance. The barplots are clustered by dataset size. Each cluster consists of six bars for different number of containers and each bar is the sum of two transfer times for stage-in and stage-out, in seconds.

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 either currently sequential or not yet in place. The wall clock times of the main steps of the data reduction pipeline as given in Table 5. The principal 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.

step	data size (GB)	execution time (s)
calibrator DI	25	8534
target DI	433	11909
init-subtract	76	37212
DD1 (DDF)	76	208800 (2d + 10h)
DD2 (FACTOR)	76	172800 (~2d)

Table 5: wall clock times of the main steps of the data reduction pipeline

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 data transfer can only happen at a later stage of 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. 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.

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

Table 6: UC4 data generation time

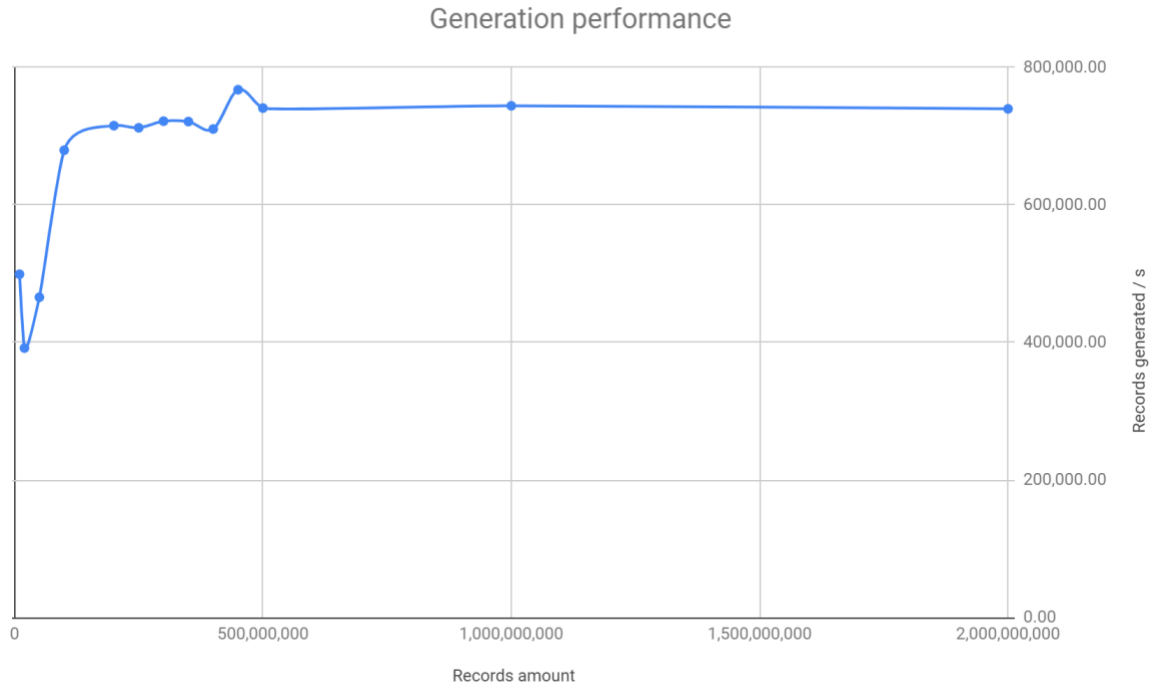

Figure 8: UC4 data transfer measurements

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 about exponentially for random forest model and linearly for deep neural network model. The biggest issue was that both the application and the H2O cluster 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.

Records amount	Model generation 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

Table 7: UC4 model generation time

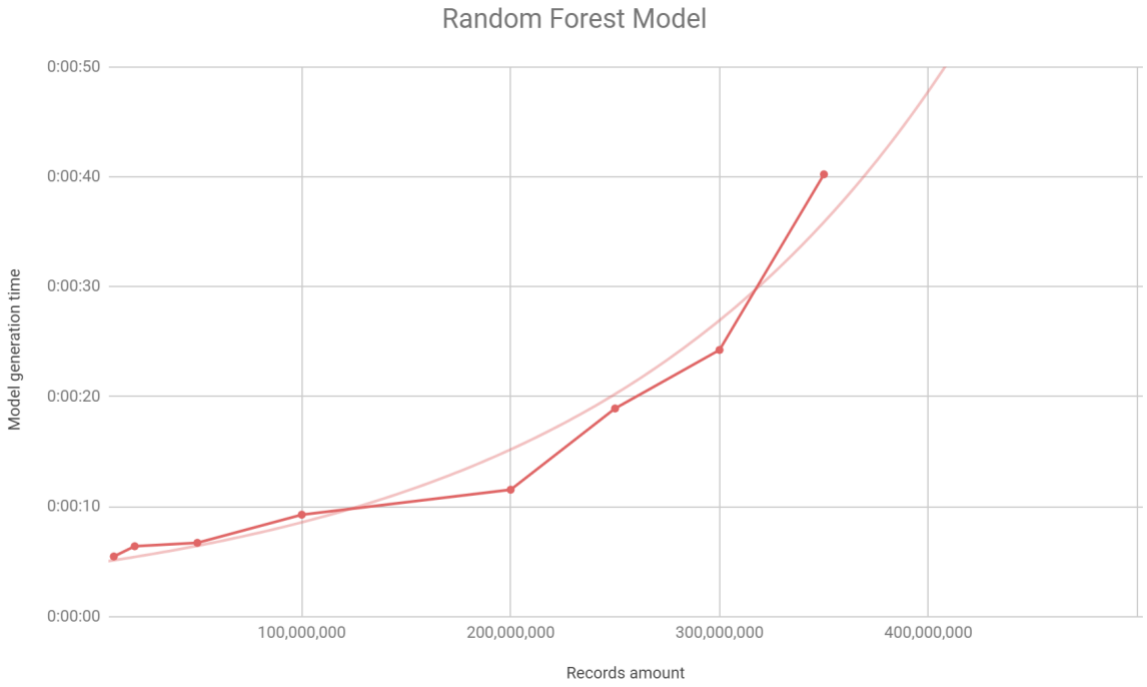

Figure 9: UC4 data transfer model

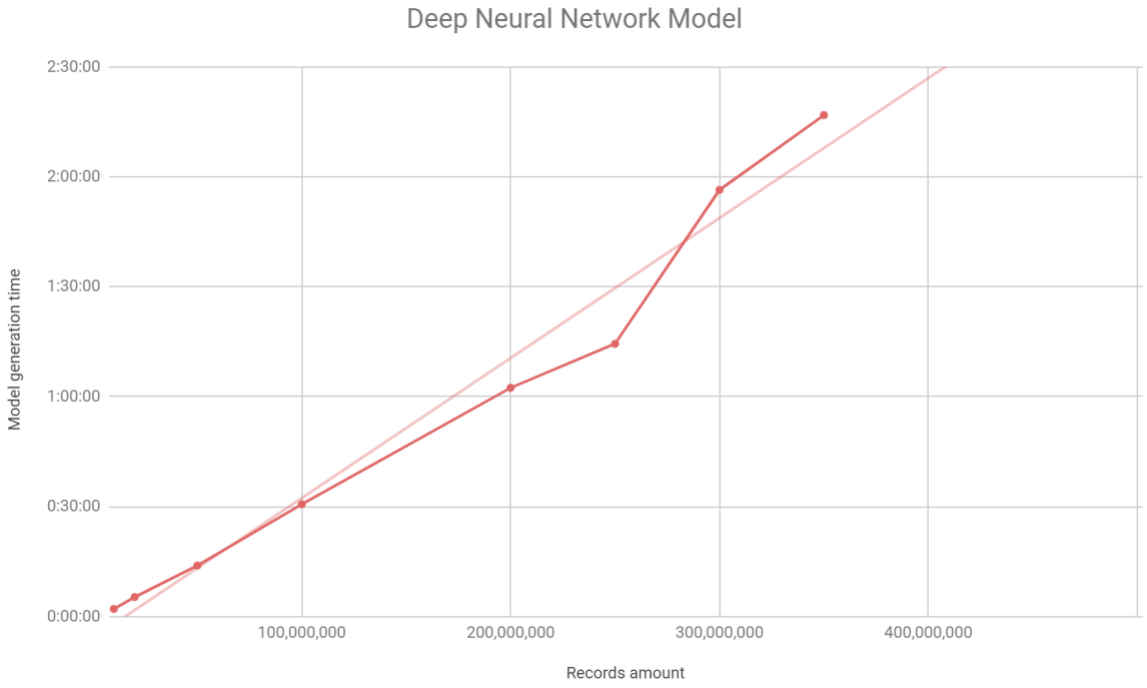

Figure 10: UC4 data generator performance

4.2.4 UC5

Use Case 5 is a closed source application, which is only started by the PROCESS IEE. Therefore, the actual execution is not controlled by PROCESS and cannot be measured. The output of Use Case 5 is not yet measured, since the application and use case owner need to clarify how the output is delivered to the end-user of the PROCESS portal. These measurements will be added to the final deliverable in D3.3.

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. Given the lack of sufficient data, we again use simple correlation analysis to get insight into the data. Modelling results for overhead of both IEE and UC2 environment are shown in Figure 11. For IEE, the linear regression in the subfigure on the left (*overhead (o) = 0.0024 * (number of containers (c)) + 2.4*) 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. Unfortunately, the very small value of the coefficient of determination R_2 measuring the goodness-of-fit suggests basically that our model is not good enough for doing prediction of future overhead. However, to put this in context, if we assume our data can be confidently fitted by such a model than the overhead of processing the entire LOFAR LTA archive (around 1800 observations of 16TB) would be only about 7s. For UC2 environment model shown in the subfigure on the right, the overhead would be much higher, 4683s (~1day), according to the regression equation *o = 2.6 * c + 3.7*. In contrary to IEE, here, the model is reliable with acceptable R_2 value (0.88) giving confidence about the predictions. Besides, one day is negligible compared to the months it would take to process all observations in the archive, each one taking four to five days to process. A final observation is that we use more data for UC2 than for IEE.

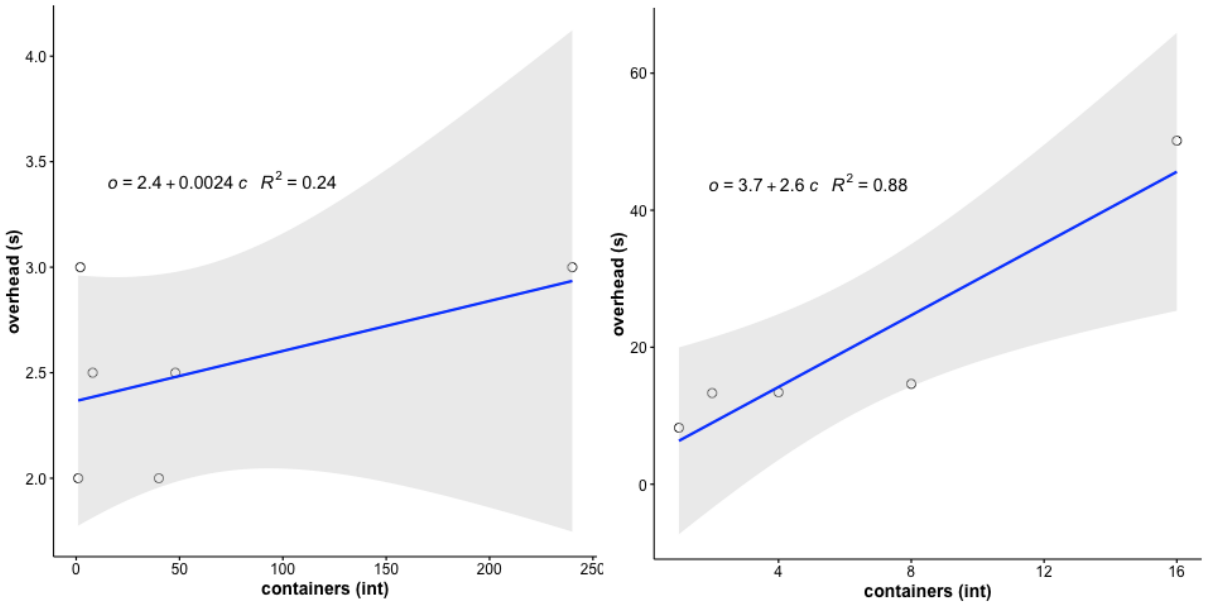

Figure 11: PROCESS overhead models.

As obviously the overhead data in IEE does not appear to have a linear relation with the number of containers, we use advanced regression analysis to dig deeper into the data. Using a random forest approach, we can fit the data with a better model, shown in Figure 12. It appears from this model that the scheduling overhead for processing all LOFAR LTA for UC2 is even lower (2.81s) although one could argue that the model is limited by the modest number of data points.

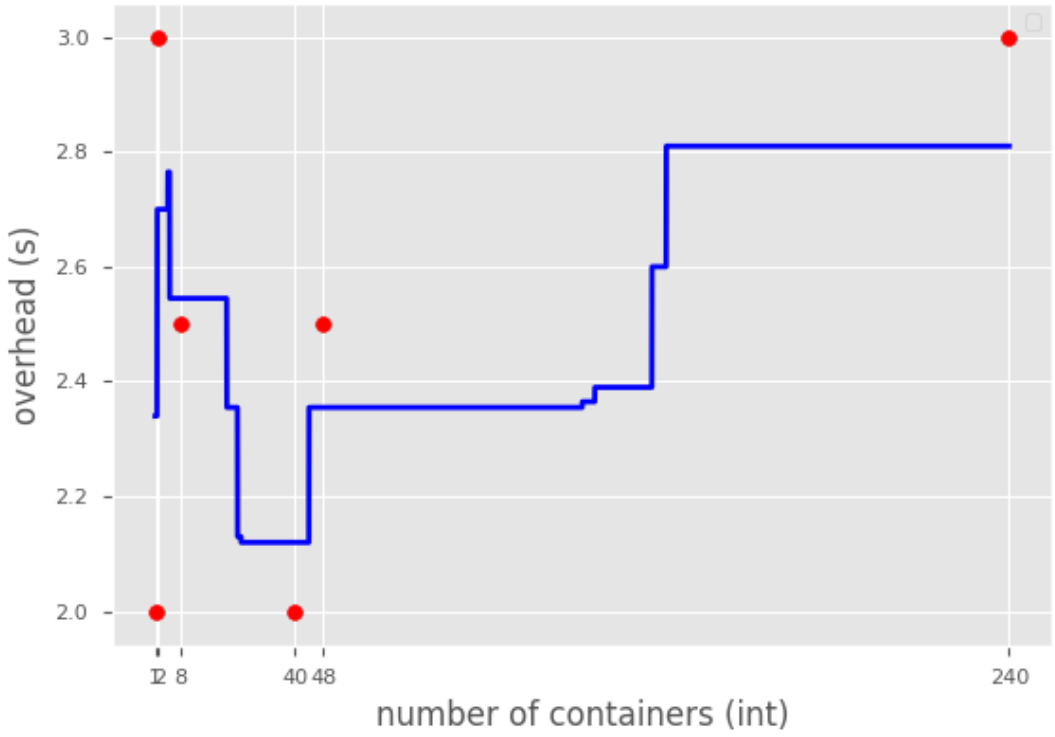

Figure 12: PROCESS overhead model with random forest regression. Actual measures are shown in red points whereas predictions are shown in blue.

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. We did our best to exclude the overhead due to WMS, but somehow it impacts the queueing delays reported above and modelled here. As illustrated in Figure 13, for both IEE (left) and UC2 environment (right), the value of the goodness-of-fit is pretty high (>= 0.90) bringing trust to the corresponding models. Again, IEE model shows a slow variation of the scheduling overhead proportionally to the number of containers ($o = 0.044 * c + 1.9$) whereas for UC2, the variations of both dimensions are of the same order ($o = c + 1.4$). The take-away message is that the scheduling due to PROCESS creates some burden, the latter is moderate and does not a show-stopper. Of course, this needs to be supported by robust models from more data.

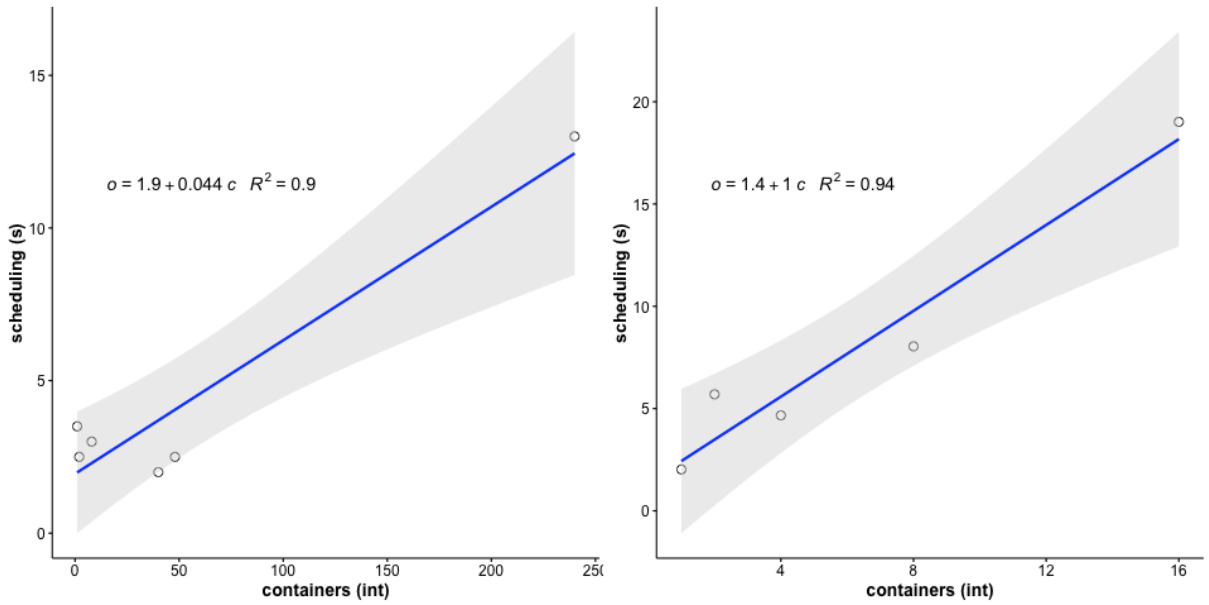

Figure 13: PROCESS scheduling overhead models in IEE (left) and UC2 (right).

5.3 Data transfer model and projection

We model and summarize data staging and transfer measurements done in this deliverable and in D8.1.

In D8.1, section 3.2, pages 13-14, we showed the data staging times in function of the size at two LOFAR LTA locations, Amsterdam, NL and Poznan, PL. Here we model the average behaviour of the staging on these two sites as illustrated in Figure 14.

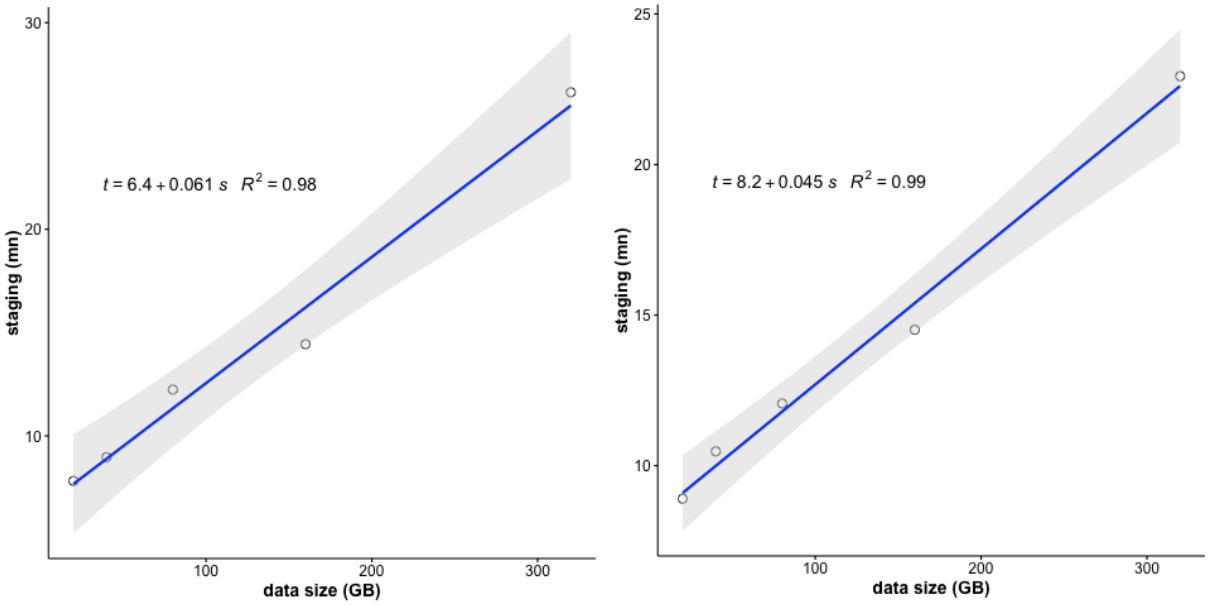

Figure 14: PROCESS data staging time models in Poznan (left) and in Poznan and Amsterdam (right). The staging time (t) in minutes) is expressed as a linear function of data size (s) in GB.

According to the average model, it would take 745.480 minutes (half a day) to stage a LOFAR observation of 16TB.

Transfer speed measurements between LOFAR LTA locations and PROCESS computing sites on one hand (D8.1), and between the computing sites on another (see section 4.2.1 above), have also been conducted. The first approach which did not use DTNs showed poor average transfer performance (5-12MB/s), whereas the second approach with DTNs shows promising transfer rates of up to 550MB/s. The latter experimented only on UC1 needs to be extended to the other use cases, for instance UC2 requiring data transfers from LTA locations to the computing sites.

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. However, this needs to be supported by more robusts models from much more data than currently.

The scheduling overhead model also shows that the scheduling due to PROCESS creates a moderate burden and does not constitute a bottleneck. Just as for the platform overhead, we need more data to confirm this trend.

Finally, the staging model shows that staging can take quite some time, especially for UC2. Unfortunately, it is beyond the control of PROCESS. In opposition, the data transfers crucial to PROCESS performance can be improved by the use of DTNs or similar approaches/protocols.

6 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.