Computing and Informatics, Vol. 39, 2020, 644–677, doi: 10.31577/cai_2020_4_644

REFERENCE EXASCALE ARCHITECTURE (EXTENDED VERSION)

Martin Bobák, Ladislav Hluchý, Ondrej Habala, Viet Tran

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

Reginald Cushing, Onno Valkering

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

Adam Belloum

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

Mara Graziani, Henning Müller

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

Souley Madougou, Jason Maassen

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

Abstract. While political commitments for building exascale systems have been made, turning these systems into platforms for a wide range of exascale applications faces several technical, organisational and skills-related challenges. The key technical challenges are related to the availability of data. While the first exascale machines are likely to be built within a single site, the input data is in many cases impossible to store within a single site. Alongside handling of extreme-large amount of data, the exascale system has to process data from different sources, support accelerated computing, handle high volume of requests per day, minimize the size of data flows, and be extensible in terms of continuously increasing data as well as an increase in parallel requests being sent. These technical challenges are addressed by the general reference exascale architecture. It is divided into three main blocks: virtualization layer, distributed virtual file system, and manager of computing resources. Its main property is modularity which is achieved by containerization at two levels: 1) application containers – containerization of scientific workflows, 2) micro-infrastructure – containerization of extreme-large data service-oriented infrastructure. The paper also presents an instantiation of the reference architecture – the architecture of the PROCESS project (PROviding Computing solutions for ExaScale ChallengeS) and discusses its relation to the reference exascale architecture. The PROCESS architecture has been used as an exascale platform within various exascale pilot applications. This paper also presents performance modelling of exascale platform with its validation[1].

Keywords: Exascale, architecture, validation

1 INTRODUCTION

New scientific instruments (e.g. distributed radio telescopes such as LOw-Frequency ARray – LOFAR, Square Kilometre Array – SKA, space telescopes such as Copernicus sentinels, etc.) are producing data at an accelerating pace. LOFAR observations are stored in the long term archive (LTA) which is distributed over Amsterdam, Jülich and Poznan. It currently contains around 30 PB of data and grows with 5 to 7 PB/year. SKA represents an even bigger challenge. It is expected that a raw

[1] This is the extended version of our paper about the reference exascale architecture [3].

data will grow by zettabytes/year which will produce 130 to 300 PB/year of correlated data. Copernicus sentinels also present an exascale challenge. They produce approximately 7.5 PB of raw data each month.

Another significant amount of data is generated by branches that are digitized. A typical example is medical science. The final report of the High Level Expert Group on Scientific Data [15] describes it as follows: "In 2010, about 2.5 petabytes – more than a million, billion data units – are stored away each year for mammograms in the US alone. World-wide, some estimate, medical images of all kinds will soon amount to 30 % of all data storage."

With the rapid growth of data [9, 8] it is often required to migrate data to a remote computation location [1]. Often the data structures are very complex and are stored in a (geographically) distributed infrastructure. Those features are so significant, that new approaches and methods need to be investigated. This paper presents an architectural blueprint that allows the whole data and compute infrastructure to draw maximal benefits from the emerging exascale capacities.

The main aim of this paper is to describe the reference architecture for exascale systems which starts from the gathering of requirements to its application in real world use cases. In the context of our work and of this paper, an exascale system is one that uses exabytes of data or exaflops of computational power. The design of the reference architecture is driven by the requirements analysis of the various use cases which come from diverse scientific communities as well as from industry. Through the generalisation of them, the reference architecture is proposed.

This paper has the following structure:

- Section 2 represents the requirements analysis of the various exascale-related use cases.
- Section 3 presents the reference exascale architecture.
- Section 4 describes the updated technology-based architecture of the PROCESS Project.

Extended version: New exascale aspects were investigated more deeply. The improvements focused on data transfer which was identified as the main bottleneck of the PROCESS platform prototype. The problem is addressed by a dedicated set of nodes – data transfer nodes. The second significant update is dedicated to the optimization of computing resources management. The second prototype of the PROCESS platform supports both cloud and HPC resources through dedicated managers Cloudify for cloud resources and Rimrock for HPC resources. Last but not least, Cloudify is successfully integrated with the European Open Science Cloud.

Meanwhile, the performance modelling and PROCESS platform validation were finished. The performance model 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 extended version has the following new sections:

- Section 5 presents the performance model.
- Section 6 describes experiments on the PROCESS platform prototype.

2 MOTIVATION

There are many research communities reaching the exascale threshold. This section investigates requirements coming from the following communities[2]:

1. medical science,
2. astronomy, and
3. ancillary pricing [6].

The requirements coming from the communities can be divided into two groups:

1. computational requirements, and
2. accessing and storing of data sets (exceeding petabytes).

The two requirements categories are not completely isolated. Contrariwise, both of them are interlaced which also creates new additional requirements. One of them is distributed and/or parallel processing of data which is the consequence of data amount generated by various simulations, and observations conducted by the above mentioned exascale research communities coming from distributed data sources and/or laboratories.

2.1 Exascale Learning on Medical Image Data

The medical use case focuses on automated cancer diagnostics and treatment planning. The aim is to study cancer detection, localisation and stage classification. Cancer diagnostics is based on the automatic analysis of a biopsy or surgical tissue specimens, which are captured by a high resolution scanner and stored in a multi-resolution pyramid structure. The size of the data set is huge (up to PBs), since it also includes tissue that is not relevant for cancer diagnosis (e.g. background, stroma, healthy tissue, etc.).

The key components of this use case are focused on pattern recognition, statistical modelling and deep learning. Thus the core requirement is to support performing dense linear algebra on distributed-memory HPC systems. Multiple GPU

[2] They are part of the PROCESS project, `http://process-project.eu/`

computation libraries should be used to merge multiple CUDA kernels. Furthermore, top-level development of the deep models should be performed using the most common machine learning and deep learning frameworks.

To process a huge amount of data (PBs and more), training needs to be distributed across different computing centres what requires automated detection of the optimal model parameters, and efficient scheduling and monitoring of processes to the available resources. The requirements analysis pointed out the need for containerization of the whole approach.

The typical requirement coming from this kind of data processing tasks is huge amount of computing power [7, 11]. Extremely large datasets might be difficult to download [14], and hospitals might require a high level of confidentiality. A generalised solution, the "Evaluation as a Service" (EaaS) could be viewed as a "clean slate" approach to deal with very large datasets, especially ones that require sophisticated access control e.g. due to privacy issues. In EaaS the data remains in a central infrastructure and does not need to be moved.

Data acquired in conjunction with hospitals needs to be pseudonymised, thus retaining a level of detail in the replaced data that should allow tracking back of the data to its original state. In this way the ethical constraints related to the usage of patient data will also be addressed.

2.2 LOFAR Use Case

Low Frequency Array (LOFAR) is a state-of-the-art radio telescope capable of wide field imaging at low frequencies. It has been ingesting data into a long-term archive (the LOFAR LTA) since 2012 and its volume is now expanding at a rate of approximately 5–7 PB/year. Its current volume is about 28 PB. This consists mostly of "Measurement Sets", i.e. visibilities – correlated signals from LOFAR stations.

The core requirement is the provision of a mechanism to run containerized workflows, thereby improving the portability and easy of use. Analysing the massive volumes of data stored in the archive is an acute problem. The environment for selecting the data and workflows has to be user-friendly, and it has to support launching the workflows, monitoring the results and downloading outputs. The whole workflow needs to be containerized by Docker or Singularity containers as workflow steps to allow each step to use different analysis tools and dependences.

The platform must have a mechanism to run the workflows on suitable processing hardware. While some parts of the workflow may run in parallel on relatively simple compute nodes (24 cores, 8 GB memory, 100 GB scratch storage), other parts currently run sequentially on a fat node with significant memory (256 GB or more, 3 TB scratch storage). The data management system has to be capable of efficiently transporting the Measurement Sets from the archive locations in Amsterdam, Jülich and Poznan to the processing locations.

The capability to horizontally scale to a significant number of compute resources to run a large number of (independent) workflows at the same time is important.

Since processing the entire archive for a single science case already requires a significant amount of core hours $O(47\,\mathrm{M})$, handling multiple science cases simultaneously will require up to exascale resources.

2.3 Ancillary Pricing for Airline Revenue Management

The ancillary pricing use case concentrates on an analysis of current ancillary sales and hidden sales pattern. This aim is tackled by machine learning approaches (e.g. random forest and neural networks) for pricing of offered ancillaries.

The core requirement is a platform that is capable of storing the incoming ancillary data in a way that allows easy exploitation for airlines. On the one hand, the platform should provide libraries for machine learning and quick processing for the model learning, while on the other hand, storing the models in an efficient way such that several hundred million ancillary pricing requests a day can be answered. The platform needs to be capable to deal with the large passenger data sets that airlines generate. It has to be based on a scalable architecture which has to foster the following features: handle large amount of data, handle data from different sources, handle a high volume of requests per day, provide quick response times, and be extensible in terms of continuously increasing data as well as an increase in parallel requests being sent.

An average airline may handle approximately 100 million passengers per year (the largest airlines carry up to twice as many passengers), each of whom will buy on average 5 ancillaries. Each ancillary record can be several kilobytes in size, and several years of data need to be processed. During processing, the size of the data is further increased due to the specifics of the used algorithms.

The data do not only need to be stored, but have to provide efficient algorithmic usage which means, on the one hand, the update of the model parameters within a reasonable timeframe (e.g. within a nightly time slot). On the other hand, this implies real-time responses with revenue-optimal prices upon customer request. Tool-stack of the platform has to support Lambda Architecture principles especially for historical data and further statistical analyses (e.g. applying mathematical and statistical algorithms on consolidated data structure to identify an optimal reference model, or applying variables of incoming requests on the optimal reference model to compute probability, estimates and forecast). The platform has to also support processing of ongoing data streams to keep the consolidated data structure up-to-date (i.e. learning new data behaviour into reference data). Also distributed computing fundamentals has to be one of the core features (e.g. support of the Hadoop ecosystem).

Ancillary data are personal data. However, they do not carry the strictest privacy requirements, since the data do not contain names, addresses or credit card information. However they may contain data that directly connect to a person such as frequent traveller information. If using real data this has to be considered as confidential information provided by the involved airlines. Therefore it has to comply with the "EU General Data Protection Regulation", if applicable. The software needs to be deployable on-site at the customer's cloud service.

3 REFERENCE EXASCALE ARCHITECTURE

After reviewing of all requirements coming from different user communities (such as medicine, radio astronomy, airline revenue management, etc.), the main challenge was to propose an architecture that is suitable for all of them. Their requirements can be divided into three main groups:

1. virtualization requirements,
2. data requirements, and
3. computing requirements.

Virtualization requirements are very straightforwardly derived from application platforms of our user communities – support of containers which offers a lightweight virtualization approach which is similar to application packages. Its advantages include easy deployment and maintenance, flexibility, reliability, scalability, etc. Since the users' applications need to be deployed on various computing infrastructures, portability and interoperability are very important features of every exascale-capable platform.

The core data requirement is handling of exascale data sets or extreme data flows which is not possible to manipulate and manage by a single data center nowadays. It brings a very demanding and ambitious request – a data federation across multiple data centers. It is also interlaced with metadata management (processing of such data is impossible without their description – the metadata). The main challenge is communication or data transmission in terms of data services. The exascale platform has to support huge data transfers across the whole infrastructure.

The main computing requirement is supporting high-performance computing as well as cloud computing which is able to offer accelerated computing also – a computing environment for the application containers. The other significant requirement is performance optimization. The current trends in the scientific applications are the following: high distribution across different research computing centers or nodes, and a degree of parallelism and concurrency also increased. Those challenges need to be taken into account during designing of computing management.

The proposed architecture is driven by modularity and scalability. These two approaches are the most suitable for an environment in which the core features are high distribution and massive parallelism. The modularity also enables to extend and adjust the platform, according to the needs of new user communities. It gives flexibility in using its sub-modules in a way which exploits the heterogeneous resources of exascale systems the most efficiently.

The aim of the proposed reference architecture is to characterize key attributes and properties that have to be handled by every scientific application using exascale data and computations. From altogether viewpoint, the reference exascale architecture (see Figure 1) is divided into the following parts (from top to bottom):

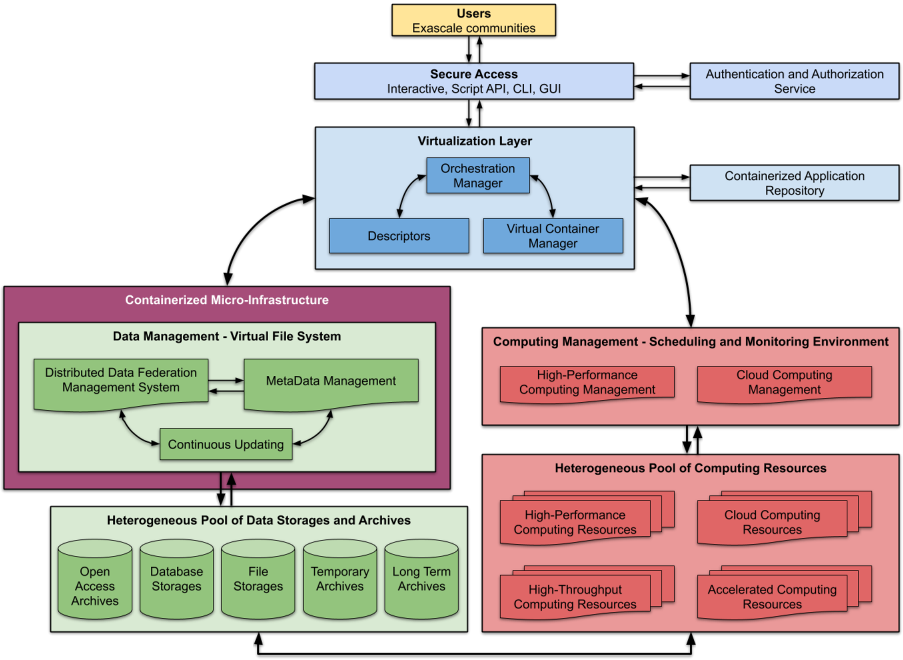

Users
Exascale communities
Secure Access
Interactive, Script API, CLI, GUI
Authentication and Authorization Service
Virtualization Layer
Orchestration Manager
Descriptors
Virtual Container Manager
Containerized Application Repository
Containerized Micro-Infrastructure
Data Management - Virtual File System
Distributed Data Federation Management System
MetaData Management
Continuous Updating
Computing Management - Scheduling and Monitoring Environment
High-Performance Computing Management
Cloud Computing Management
Heterogeneous Pool of Computing Resources
High-Performance Computing Resources
Cloud Computing Resources
High-Throughput Computing Resources
Accelerated Computing Resources
Heterogeneous Pool of Data Storages and Archives
Open Access Archives
Database Storages
File Storages
Temporary Archives
Long Term Archives

Figure 1. Reference exascale architecture

Users of the scientific exascale applications (in yellow) – the exascale system has to support functionalities required by its user communities. That also means to support legacy applications in some cases (see the Copernicus use case). According to the initial and updated requirements analysis, the best way is to build it on containerization. All of the applications are stored in a containerized repository which is available to user communities. The users are accessing the exascale platform through virtualized scientific portals which are also containers providing a user friendly graphical interface. The proposed GUI is easily templated and so modified according to the needs of its user community. The containerization approach is flexible, scalable, reusable and ready to use. Moreover, it does not require any special technical skills (especially, related to integration – exascale data processing is often contingent on complex software tools involving expert knowledge about its management) to make it run on the resource infrastructure (see the LOFAR use case).

Virtualization layer (in blue) – is situated between the containerized application repository and platform infrastructure managers. Interoperability of data and computing infrastructure is the key and critical requirement of the exascale systems. To use both infrastructures in the most efficient way, we propose the exascale reference architecture based on containerisation instead of virtual machines. The performance of the infrastructure as the whole is utilized in a better way. It is caused by minimization of overheads (e.g. software duplications). Thus virtualization layer based on containerization approach exploits the infrastructure resources in optimal way and also it supports the requirements from our user communities. The main requirement coming from the users is supporting of various application containers. According to a type of computing resources, they can be divided into two groups:

1. HPC containers, and
2. cloud containers.

Thus the virtualization layer has to be capable to handle them in cooperation with lower layers (data management and computing management).

Data management (in green) – requirements coming from the exascale scientific applications could be divided into two main groups: distributed data federation, and metadata. It is very common that the exascale scientific applications have highly complicated datasets that need to be handled and processed by relevant systems. For that purpose, its file systems must have a module capable to work with metadata. The metadata module has to be federated and distributed as well as the management system for the data infrastructure itself. At this level of the infrastructure, the system architect has to be careful whether the component will be containerized, or not. On the one hand, the exascale system has to avoid overhead and latency (according to our experiments, it is caused by needless duplication of software) thus we prefer containers to virtual machines. However, on the other hand, all infrastructural services do not need to be virtualized. For example, virtualization of HBase (through containers, or virtual machines) is not necessary because it leads to dataset duplication in the worst case or a performance overhead in the best case. Thus a better approach is to support infrastructural services ecosystem through micro-services. Micro-services serve as adapters and connectors to infrastructural services. They are integrated into a containerized micro-infrastructure, which is customized according to requirements coming from a use case and connecting them to a distributed virtual file system. The micro-infrastructure allows for application-defined infrastructures with the main advantages being threefold: First, services can be customized for the application; e.g., data staging service. Second, minimizing global state management (a major scaling issue); e.g., instead of having one global index for all files for all applications, have micro-infrastructures manage their own local indices and states. Third, micro-infrastructures are isolated from each other, which increases security between users of different applications. The PROCESS distributed file system layer needs to be virtualized because it has to run on top of multiple file systems. Also, it is crucial that access to a data storage federation is unified. Thus the virtual file system is distributed.

Computing Management (in red) – this part of the infrastructure is related to scheduling and monitoring computing resources. The infrastructure has to be loaded as balanced as possible. Two kinds of resources was recognized as suitable for exascale scientific applications by our user communities, namely: high performance computing (HPC) resources, and cloud resources. HPC manager is based on a queuing approach. Manager of cloud resources is based on REST API. Both types of resources are often enriched by support from high-throughput resources or accelerated resources. For example, GPU utilization within machine learning and deep learning application is very commonly required by those user communities, however, the requirement is still quite hard to satisfy. Since clusters build on CPUs can be used by every community, big clusters with strong GPUs are not very common nowadays.

4 PROCESS ARCHITECTURE – AN EXAMPLE OF A TECHNICAL EXASCALE ARCHITECTURE

The PROCESS project is one of exascale research projects funded by the European Union's Horizon 2020 research and innovation programme. One of its main outputs will be a modular software platform which will be capable to handle exascale challenges coming from both scientific communities as well as industry.

The PROCESS architecture shows how the exascale platforms look in the real world. It was introduced in [6] and extended by a micro-services approach which is described in the following text. Micro-infrastructure[3] is a very specialized and autonomous set of services and adaptors which interact across the extreme large data service-oriented infrastructure. Alongside the efficiency mentioned above, the approach supports scalability, high adaptability, modularity, and straightforward integration with the virtual layer. Since each use case has its own requirements and dependences, modularity together with high adaptability are very important and useful properties of every exascale environment.

The architecture is capable to support portals of external communities via REST API. The LOFAR community is driving the development one of the PROCESS use cases. The Netherlands eScience center extended the actual LOFAR portal towards a usage in the EOSC context. Therefore, besides the selection of an observation also a submission system was integrated.

This submission system connects via an API also to the PROCESS IEE, the central part of all deployments to Cloud and HPC resources done by PROCESS. To enable this communication, the PROCESS architecture had to be extended by an IEE adapter to external sources (see Figure 2). Thereby, the IEE is able to list all available pipelines defined for the LOFAR computations and deploy the entire workflow.

[3] The set of integrated micro-services.

As it can be seen, an astronomer uses the LOFAR portal as the entry point, where an observation and configuration parameters are defined. Inside the portal, the computation is started and will trigger a submission of the workflow via IEE, which calls LOBCDER to stage in the data, deploy the container and makes the output available. The addition of the API interface in the architecture was necessary, since the LOFAR community is used to the existing portal. Therefore, to not switch the user interface, the actual deployment is abstracted from the astronomers.

The data services expose interaction points through a REST management API where users can manage their private micro-infrastructure and a set of external infrastructure endpoints such as WebDAV. The authorization to the web services is ensured through tokens. Generation of the access tokens is achieved through a global access and authorization service.

The PROCESS platform has dedicated set of nodes for data transfers – data transfer nodes (DTNs). DTNs are special hardware nodes that are dedicated to the transfer of data. Such nodes have high network bandwidth and sizable storage that can be used as caches to transfer data at high speed between DTNs. Tuning of these nodes and their connections often requires network experts. In our architecture, we assume DTNs are available, pre-optimized and have some means to be programmed e.g. having the ability to deploy containers onto the DTN. LOBCDER's role is to integrate DTNs through this programmability and be able to copy data to/from DTNs.

Another typical characteristic of the exascale environment is handling of different elements for processing, distribution, and management, which requires specific hardware, or nodes. These requests are possible to satisfy by the micro-infrastructure composed of dedicated nodes, or services addressing a particular request. Since the requirements are handled by virtualization typically (abstracting details of the hardware infrastructure, or the software stack), and so the micro-infrastructure offers a natural solution.

Figure 2 depicts the changes of the initial PROCESS architecture [6, 2] needed to involve the micro-infrastructure approach into the initial architecture. All the changes are highlighted in magenta. The main change is a new way of accessing data sources (through data adapters). The described approach also simplifies it. The new version has one "branch" instead of two "branches" (one dedicated to pre/post processing tools; e.g., DISPEL, and the other dedicated to pure data access through the distributed virtual file system; e.g., LOBCDER). It also influences IEE (Jupyter is a part of micro-infrastructure, thus the IEE needs only a plugin for it), and LOBCDER (the data infrastructure management layer responsible for integration of lower adjacent tools was added).

The PROCESS architecture is also a result of applying the reference exascale architecture which represents the common features of the PROCESS platform (as well as every exascale-related platform, or application). On its top users are interacting with the platform through a secure access. IEE represents the environment for users, however, security is out of the project scope. Therefore, this aspect is not inves-

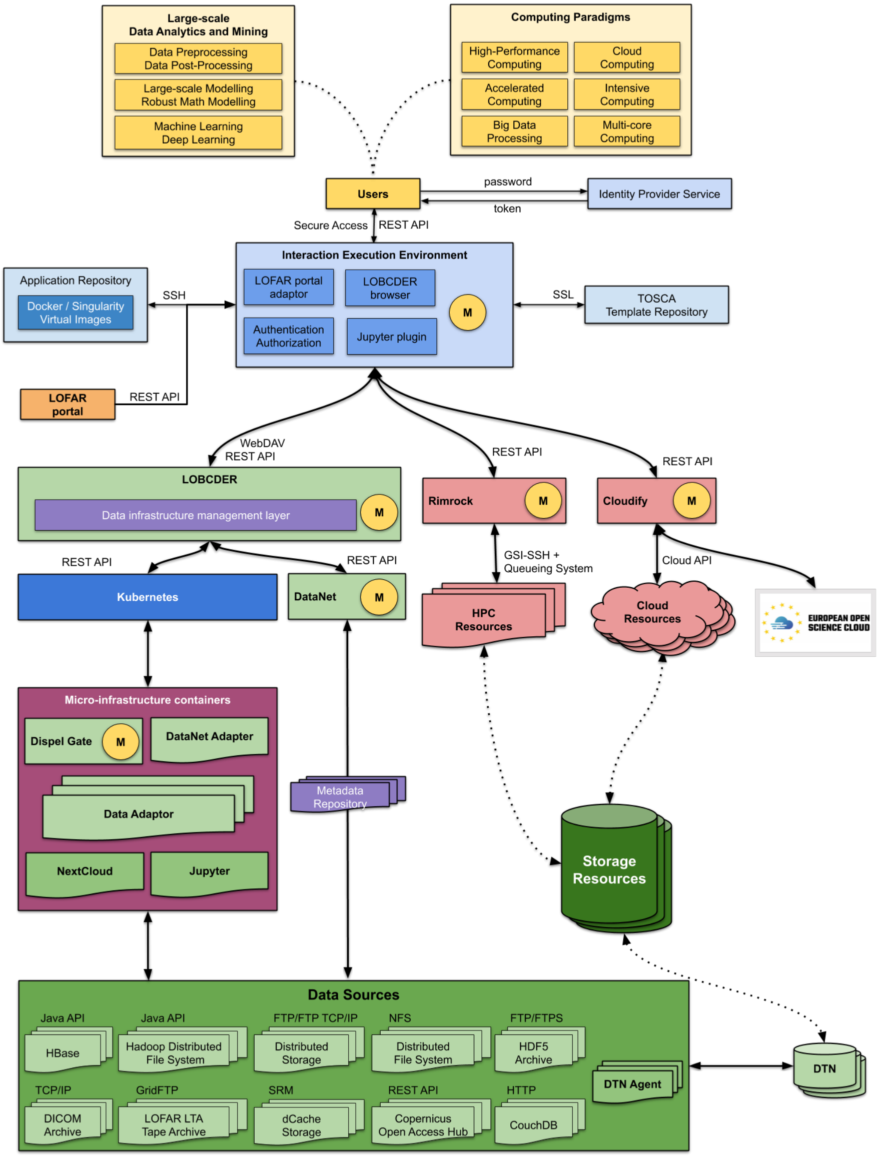


Figure 2. PROCESS architecture. (The yellow “M” token represents connection with a monitoring system. The PROCESS platform uses Zabbix as a monitoring agent.)

tigated anymore. The whole resource infrastructure is orchestrated by the virtual layer. Below that layer a virtual file system alongside HPC and Cloud managers is situated. The virtual file system is containerized through micro-infrastructure. The main reason behind this decision is that the use cases have different requirements (e.g. need to access various data sources). Micro-infrastructure containers are managed by Kubernetes. Last but not least, HPC and Cloud managers. Both of them have to be scheduled and users have to have information coming from monitoring tools about their task as well as raw hardware infrastructure. Rimrock is used as a unified environment for managing HPC resources, and Cloudify[4] for managing of cloud resources.

Cloudify can deploy VMs on any sites with OpenStack[5] middleware, including sites in EOSC-Hub Federated Cloud infrastructure[6]. It opens the door for users to access computation resources and integration with the EOSC. For the full execution of use cases on EOSC Federated Cloud, other components should be integrated with EOSC-Hub, too, mainly user portal and data infrastructure.

5 PERFORMANCE MODELLING

Proposed performance model is 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, we use these results to create an analytical model that will allow us to extrapolate the performance based on given measurements.

The performance model is based on generic performance modelling techniques and a classification [5][7] developed within CRESTA project[8]. Table 1 defines four main categories varying from raw measurements, over benchmarking and simulations to complex analytical modelling with a large number of parameters.

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 is used to identify performance bottlenecks in the architecture and assists in debugging and verifying its correctness. The microbenchmark

[4] Network Orchestration & Edge Networking — Cloudify: `https://cloudify.co/`.

[5] Open Source Cloud Computing Infrastructure – OpenStack: `https://www.openstack.org/`.

[6] EGI Cloud compute: `https://eosc-hub.eu/services/EGICloudCompute`.

[7] David Henty: Performance Modelling [online: `https://materials.prace-ri.eu/499/9/Performance_modelling.pdf`]

[8] CRESTA project (Collaborative Research Into Exascale Systemware, Tools and Applications) homepage: `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 taken from David Henty: Performance Modelling [online: `https://materials.prace-ri.eu/499/9/Performance_modelling.pdf`] [5]

is a very simple application (validation or test pipeline) running through the complete architecture and gathering first results.

Simulation and Analytical Modelling: Executing and measuring a given application running on the proposed platform 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.

5.1 Identification of Measurands

We stress that the hardware infrastructure such as computing, storage, and network have a big impact on the performance of services. However, 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 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 paper. The data transfers are hidden by overlapping data transfer with computing or use pre-fetching and caching to minimize the data transfers.

Based on the use case requirements analysis, 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.

T2: Deployment Strategy – Part of T2 is the time needed to decide on which computing and storage sites 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 the platform 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 is affected by each compute site's scheduling system that is not in the scope of the paper. 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 storage infrastructure. Based on the amount of data and the specified workflow the data service needs to choose a suitable stage-out strategy.

[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`

T8: Stage-Out – With the appropriate stage-out strategy the output data now needs to be transferred to the chosen storage resource.

Figure 3 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:

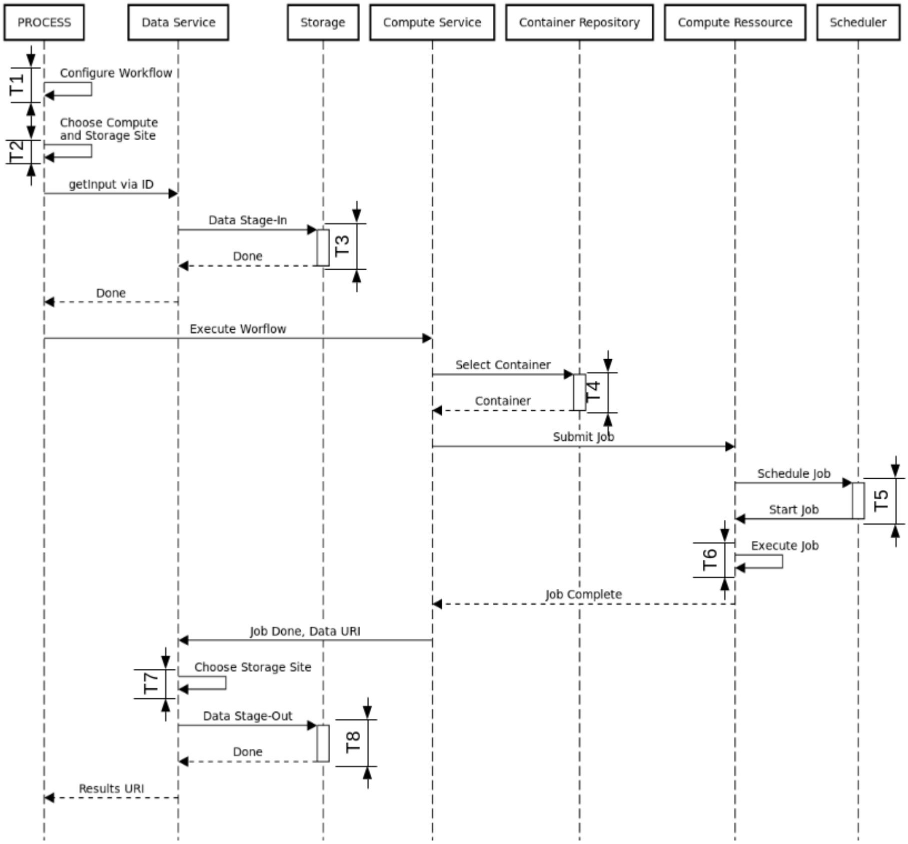


Figure 3. Sequence diagram describing the steps involved in execution of a typical application scenario

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

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

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 our measurands

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 4 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 4 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 4 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.

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.

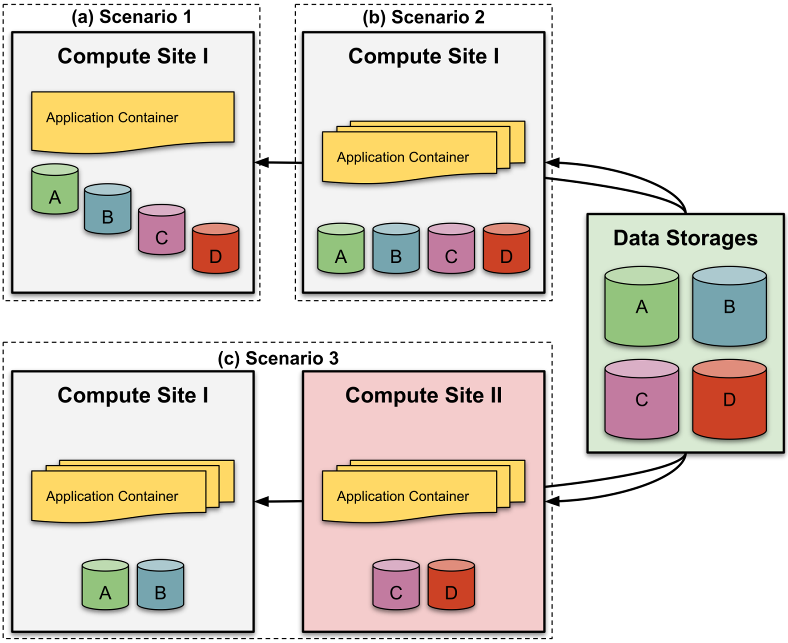


Figure 4. Three measurement scenarios: a) Single container – single site, b) Multiple container – single site, c) Multiple container – multiple site. In all three scenarios Stage-In and Stage-Out will downscale the system overall performances, unless we address the data transfer over a wide area network.

5.2 Runtime Composition

Based on Figure 4 the total runtime of an application can be defined as follows:

$$\begin{aligned} Runtime &= Overhead + DataTransfer + Scheduling + ExecutionTime, \\ Overhead &= T1 + T2 + T4 + T7, \\ DataTransfer &= T3 + T8, \\ Scheduling &= T5, \\ ExecutionTime &= T6. \end{aligned}$$

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 *DataTransfer*, *Scheduling* and *ExecutionTime* 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, to have 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 *DataTransfer* 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, [10] 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, [12] 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 of 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, [13] 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 [4], a similar technique is used to fine tune the execution time estimate provided by the user.

5.3 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.

5.4 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.

6 EXPERIMENTAL RESULTS

6.1 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. 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 5 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.

In Figure 6 a) (left), the measured values of T2 are grouped by input data size of 10 MB, 100 MB, 1 GB and 10 GB which are the test input data sizes. We observe that up to 1 GB, the average value of T2 is quite small with little deviation from that value. However, for the 10 GB 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 6 b) (right). We can indeed observe that the average submission delays are close within the 2–3 seconds range. The only reason the 10 G batch is larger is because of an outlier at container count 96 for this 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

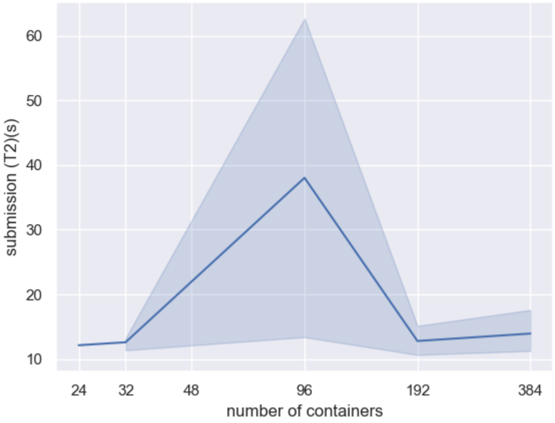


Figure 5. Submission delay (T2) behaviour in the PROCESS production prototype

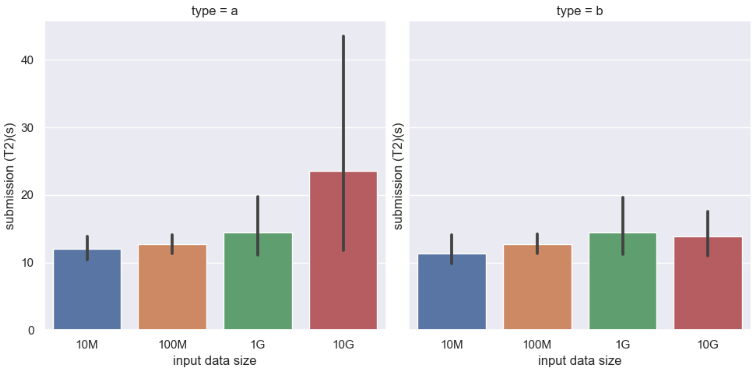


Figure 6. Submission delay in IEE batched by the input data size

the only one we will consider here. The behaviour of the metric is shown in Figure 7 below. We observe that the implicit staging is of the same order of magnitude as T2 but at a generally lower scale.

The cumulative behaviour of T2 and implicit staging is illustrated in the Figure 8 below. The principal observation is that the global overhead is moderate. It culminates at 25 s at container count 384.

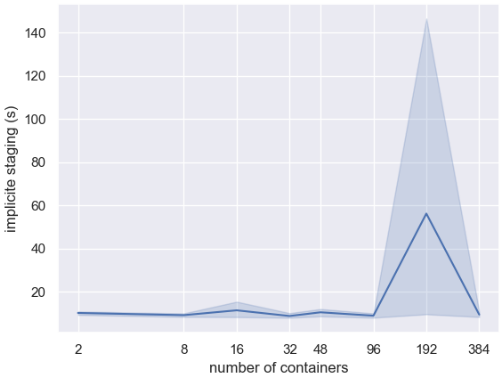


Figure 7. Implicit staging overhead behaviour in the PROCESS production prototype

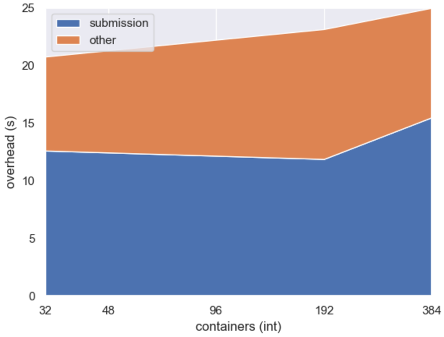


Figure 8. Overall overhead (T2+implicit staging) behaviour in the PROCESS production prototype

6.2 Scheduling Measurements

Overhead due to scheduling in IEE is measured as queueing times which are plotted in Figure 9 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.

A complementary explanation of the scheduling behaviours is given in Figure 10. Just as Figure 9 shows that T5 does not depend on the container count, Figure 10 shows it does not depend on the input data size neither. The delay batches for 1 GB

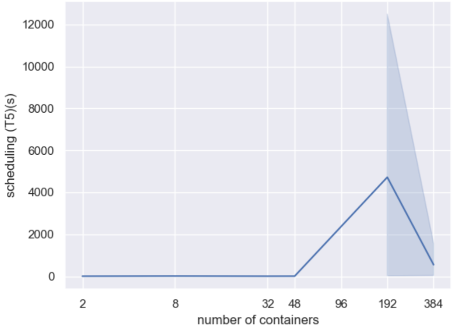


Figure 9. PROCESS scheduling overhead measurements from IEE

and 10 GB are both lower than that of 100 M, which is where the abovementioned outlier happens to be.

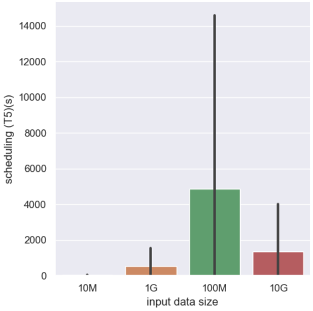


Figure 10. Scheduling delay measurements in the PROCESS production prototype batched by input size

6.3 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 the PROCESS data services, which we report in Figure 11. 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 1 GB, the transfer duration for 10 GB clearly increases. This is expected as transfer time only depends on the input size and network performance and conditions. The latter is probably responsible for the important spread at container count 192.

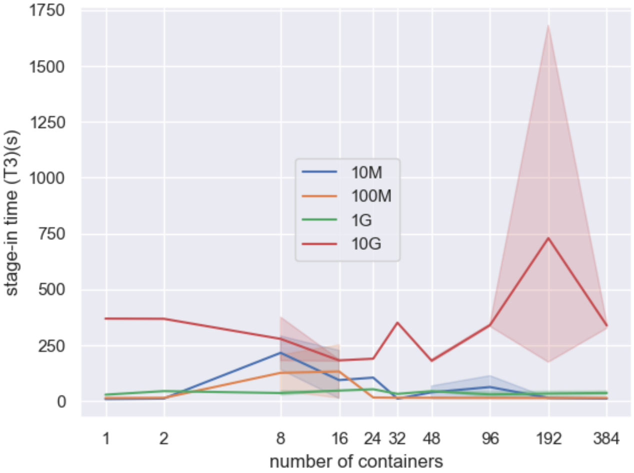


Figure 11. Staging-in measurements in the IEE production prototype

6.4 Overhead Model and Projection

Using the collected measurement data, 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 12. The linear model (equation: $\mathit{overhead}\ (o) = 0.011 * (\mathit{number\ of\ containers}\ (c)) + 20)$ 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 1 800 observations of 16 TB) would only be about 40 seconds.

Figure 12 a) (left) plots the residual values for each observation and allows to check whether the regression model is appropriate for the dataset. If it is, then the values should be randomly scattered around the value $y = 0$. As this is what we observe in Figure 12 b) (right), we are confident that our approach is appropriate.

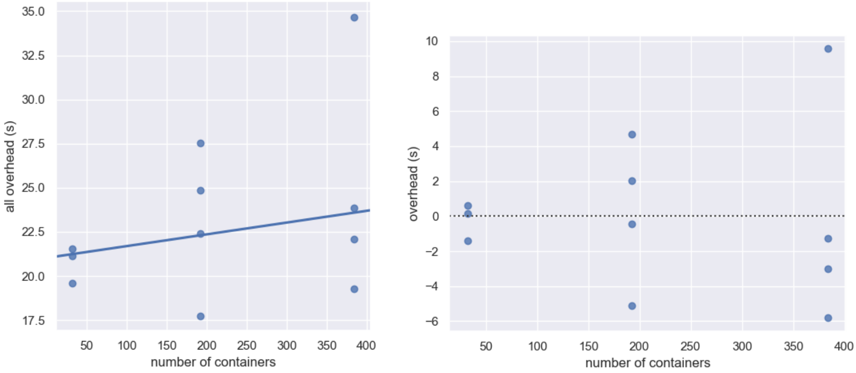


Figure 12. PROCESS T2 overhead models

6.5 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 13, 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.

6.6 Data Transfer Model and Projection

In Section 6.3, 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 14.

We know that the staging (in and out) performance depends instead on the input data size whose linear model is shown below in Figure 15. The equation of the shown linear model is $T3 = 0.032 * \mathrm{M} + 39$ where M is the size of the input data in MB.

According to the linear model, it would take on average about 359 s to transfer 10 GB of data which makes for an average speed of about 27.85 MB/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 16 TB.

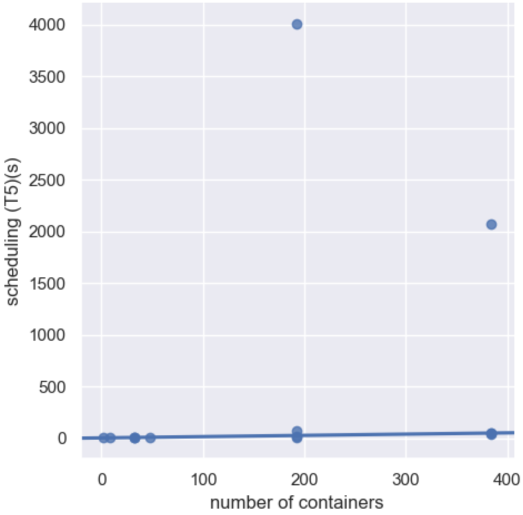


Figure 13. PROCESS scheduling overhead models in the IEE production prototype

7 CONCLUSION

This paper presents the motivation for the exascale architecture coming from PROCESS use cases. The medical use case represents a computationally intensive application requiring accelerated computing. The LOFAR use case requires highly

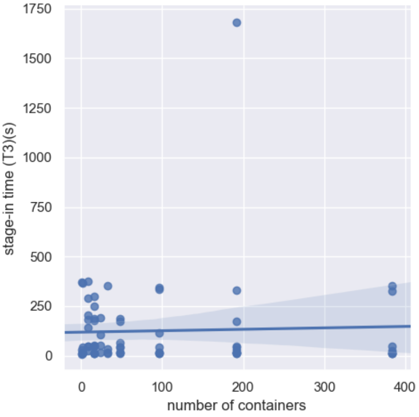


Figure 14. PROCESS staging-in delay model in IEE

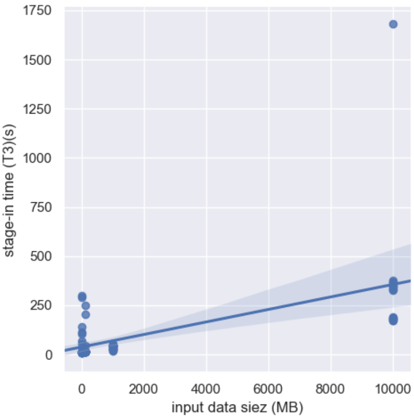


Figure 15. PROCESS staging-in delay model in IEE

effective exascale workflows for exascale datasets. The ancillary pricing use case embodies exascale throughput requirements.

The reference architecture combines several design approaches (e.g. modularity, service-oriented architectures) with computational paradigms (e.g. high-performance computing, cloud computing, accelerated computing). Consequently, the reference architecture is used within the PROCESS project as a blueprint for the PROCESS architecture.

The paper 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. 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.

Acknowledgment

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

REFERENCES

[1] Ashby, S.—Beckman, P.—Chen, J.—Colella, P.—Collins, B.—Crawford, D.—Dongarra, J.—Kothe, D.—Lusk, R.—Messina, P. et al.: The Opportunities and Challenges of Exascale Computing-Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee. U.S. Department of Energy, Office of Science, 2010.

[2] Bobák, M.—Belloum, A. S. Z.—Nowakowski, P.—Meizner, J.—Bubak, M.—Heikkurinen, M.—Habala, O.—Hluchý, L.: Exascale Computing and Data Architectures for Brownfield Applications. 2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), Huangshan, China, IEEE, 2018, pp. 461–468, doi: 10.1109/FSKD.2018.8686900.

[3] Bobák, M.—Hluchý, L.—-Belloum, A.—Cushing, R.—Meizner, J.—Nowakowski, P.—Tran, V.—Habala, O.—Maassen, J.—Somosköi, B. et al.: Reference Exascale Architecture. 2019 15th International Conference on eScience (eScience), San Diego, CA, USA, IEEE, 2019, pp. 479–487, doi: 10.1109/eScience.2019.00063.

[4] Gaussier, E.—Glesser, D.—Reis, V.—Trystram, D.: Improving Backfilling by Using Machine Learning to Predict Running Times. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15), Austin, TX, USA, IEEE, 2015, pp. 1–10, doi: 10.1145/2807591.2807646.

[5] Henty, D.—Holmes, D.—Booth, S.—Bethune, I.—Gibb, G.—Reid, F.: Writing Scalable Parallel Applications with MPI @ EPCC 2017. 2016.

[6] Hluchý, L.—Bobák, M.—Müller, H.—Graziani, M.—Maassen, J.—Spreeuw, H.—Heikkurinen, M.—Pancake-Steeg, J.—Spahr, S.—vor dem Gentschen Felde, N. O.—Höb, M.—Schmidt, J.—Belloum, A. S. Z.—Cushing, R.—Nowakowski, P.—Meizner, J.—Rycerz, K.—Wilk, B.—Bubak, M.—Habala, O.—Šeleng, M.—

Dlugolinský, S.—Tran, V.—Nguyen, G.: Heterogeneous Exascale Computing. In: Kovács, L., Haidegger, T., Szakál, A. (Eds.): Recent Advances in Intelligent Engineering. Chapter 5. Springer, Cham, Topics in Intelligent Engineering and Informatics, Vol. 14, 2020, pp. 81–110, doi: 10.1007/978-3-030-14350-3_5.

[7] Jimenez-del-Toro, O.—Otálora, S.—Andersson, M.—Eurén, K.—Hedlund, M.—Rousson, M.—Müller, H.—Atzori, M.: Analysis of Histopathology Images: From Traditional Machine Learning to Deep Learning. Chapter 10. Biomedical Texture Analysis: Fundamentals, Tools and Challenges. Elsevier, 2017, pp. 281–314, doi: 10.1016/B978-0-12-812133-7.00010-7.

[8] Kune, R.—Konugurthi, P. K.—Agarwal, A.—Chillarige, R. R.—Buyya, R.: The Anatomy of Big Data Computing. Software: Practice and Experience, Vol. 46, 2016, No. 1, pp. 79–105, doi: 10.1002/spe.2374.

[9] Lee, J.-G.—Kang, M.: Geospatial Big Data: Challenges and Opportunities. Big Data Research, Vol. 2, 2015, No. 2, pp. 74–81, doi: 10.1016/j.bdr.2015.01.003.

[10] Liu, Z.—Balaprakash, P.—Kettimuthu, R.—Foster, I.: Explaining Wide Area Data Transfer Performance. Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '17), 2017, pp. 167–178, doi: 10.1145/3078597.3078605.

[11] Montavon, G.—Samek, W.—Müller, K.-R.: Methods for Interpreting and Understanding Deep Neural Networks. Digital Signal Processing, Vol. 73, 2018, pp. 1–15, doi: 10.1016/j.dsp.2017.10.011.

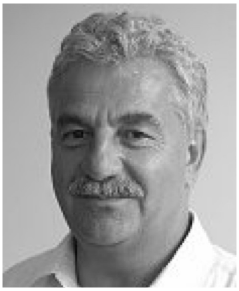
[12] Nurmi, D.—Brevik, J.—Wolski, R.: QBETS: Queue Bounds Estimation from Time Series. In: Frachtenberg, E., Schwiegelshohn, U. (Eds.): Job Scheduling Strategies for Parallel Processing (JSSPP 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4942, 2007, pp. 76–101, doi: 10.1007/978-3-540-78699-3_5.

[13] Smith, W.—Foster, I.—Taylor, V.: Predicting Application Run Times Using Historical Information. In: Feitelson, D. G., Rudolph, L. (Eds.): Job Scheduling Strategies for Parallel Processing (JSSPP 1998). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1459, 1998, pp. 122–142, doi: 10.1007/BFb0053984.

[14] Szegedy, C.—Liu, W.—Jia, Y.—Sermanet, P.—Reed, S.—Anguelov, D.—Erhan, D.—Vanhoucke, V.—Rabinovich, A.: Going Deeper with Convolutions. Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9, doi: 10.1109/CVPR.2015.7298594.

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

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



Ladislav Hluchý is the Head of the Parallel and Distributed Information Processing Department, the Vice-Director of the Institute of Informatics, Slovak Academy of Sciences (IISAS). He received his M.Sc. and Ph.D. degrees, both in the computer science. He is R & D project manager, work-package leader and coordinator in a number of 4th, 5th, 6th, 7th and H2020 EU IST RTD projects as well as Slovak national projects. His research topics are focused on parallel and distributed computing, large scale applications, cluster/grid/cloud computing, service oriented computing and knowledge oriented technology. His highlighted research works are within EU IST RTD projects PROCESS H2020-777533 PROviding Computing solutions for ExaScale challengeS, DEEP-HybridDataCloud H2020-777435, EOSC-Synergy H2020-857647, EOSC-hub H2020-777536, EGI-Engage H2020-654142, EGI-InSPIRE FP7-261323, EGEE III FP7-222667, EGEE II FP6 RI-031688, EGEE FP6 INFSO-RI-508833, REDIRNET FP7-607768, VENIS FP7-284984, SeCriCom FP7-218123, Commius FP7-213876, ADMIRE FP7-215024, DEGREE FP6-034619, INTAS FP6 06-1000024-9154, int.eu.grid FP6 RI-031857, K-Wf Grid FP6-511385, MEDIGRID FP6 GOCE-CT-2003-004044, CROSSGRID FP5 IST-2001-32243, PELLUCID FP5 IST-2001-34519, ANFAS FP5 IST-1999-11676, SEIHPC, SEPP and HPCTI. He is a member of IEEE, e-IRG, EGI Council, the Editor-in-Chief of the Current Contents journal Computing and Informatics (CAI). He is also (co-)author of nearly 600 scientific papers, contributions and invited lectures at international scientific conferences and workshops. He is a supervisor and consultant for Ph.D. study at the Slovak University of Technology (STU) in Bratislava, the Comenius University (UK) in Bratislava, and the Technical University of Košice (TUKE), Slovakia.

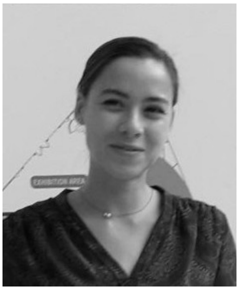
Ondrej Habala is Researcher at the Institute of Informatics, Slovak Academy of Sciences, Bratislava, Slovakia. He has been working in the Department of Parallel and Distributed Information Processing since 2001. His interests are mainly in data and metadata management in distributed computing, as well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

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

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

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

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



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

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

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

Jason Maassen is Technology Lead at the Netherlands eScience Center. He is involved in many of the projects at the center which apply parallel and distributed programming to scientific applications, ranging from high resolution climate modeling to digital forensics. In addition, he guides internal software development at the center and scouts for new software technology that can be used in projects. In the past, he participated in many research projects, such as EU FP5 GridLab, the Dutch Virtual Labs for eScience, StarPlane, PROMM-GRID, COMMIT, and H2020 PROCESS, where he worked on a range of topics related to large scale distributed computing.