

PREFACE TO THE SPECIAL ISSUE ON PROVIDING COMPUTING SOLUTIONS FOR EXASCALE CHALLENGES

Dieter KRANZLMÜLLER

Munich Network Management Team (MNM-Team)

Ludwig-Maximilians-Universität

Munich, Germany

✉

Leibniz Supercomputing Centre (LRZ)

Bavarian Academy of Sciences and Humanities

Garching, Germany

e-mail: kranzlm@mn-team.org

Maximilian HÖB

Munich Network Management Team (MNM-Team)

Ludwig-Maximilians-Universität

Munich, Germany

e-mail: hoeb@mn-team.org

Today, as we expect the appearance of the first supercomputer with exascale capabilities, some tough questions remain to be solved. A main challenge of the upcoming exascale era will appear around the ability to process extremely large datasets of several peta- and exabytes effectively and efficiently. This needs to be addressed with new management and architectural approaches, focusing on modular, integrated and light-weight solutions, which enable end-users across all scientific domains to utilize these future supercomputing systems. This special issue gives an overview on approaches focusing of these challenges and exacting applications, requiring exascale-ready compute and data services.

In their paper Bobák et al. [2] present an exascale reference architecture as a layered framework to overcome exascale challenges, developed within the PROCESS project¹. Their modular and service-oriented approach enables low-overhead

¹ www.process-project.eu

usage of the underlying HPC, cloud or accelerated-based infrastructure, completely abstracted to end-users and simplifying the configuration, deployment and data-handling process.

Data management will be a major part of this work – the building block of any successful ecosystems. All services about processing multiple sources of distributed data must be smart, since more and more data sets are geographically spread across also various administrative domains. Cushing et al. [5] propose a scalable, programmable and easy to deploy data infrastructure supporting data-intensive applications.

Together with data management, future systems need to implement efficient deployment and computing services. Meizner et al. [10] detail their scalable computing platform, enabling complex workflow deployments on heterogeneous systems. Their evaluation includes data intensive applications, showing the ability of the approach. One of those applications is based on observation data from the LOFAR radio telescope. Madougou et al. [9] describe their simple point-and-click-reduction of the complex workflow to compute calibrated sky maps out of the LOFAR observations. This workflow relies on efficient data movement and makes use of the data infrastructure presented above.

Based on extremely large digital images of tumor tissue, histopathology evolved to a computational and storage demanding application. Graziani et al. [7] present a modular, containerized pipeline for the detection of tumor regions in digital specimens of breast lymph nodes with deep learning models. The three independent layers of this pipeline were evaluated on different computing resources accessible through the services described in [5] and [10].

Having a much deeper dive into HPC systems, supercomputers face not only efficiency challenges across different systems, but also inside single clusters. Parallel applications demand efficient distributed memory access to achieve large-scale performance. To decrease the complexity of implementing scientific applications, Gschwandtner et al. [8] propose a programming interface with the ease of shared memory programming models.

As Data analysis gains more importance as more data is available while today's systems are capable of processing more data. This can be seen not only on HPC clusters, but also on cloud resources. To assist end-users building scientific, cloud-based data analytic pipelines, Baranowski et al. [3] present the Cookery framework.

Bystrov et al. [4] investigate the performance of haemodynamic flow computations on a cloud infrastructure, focusing on the parallel performance analysis, energy consumption and virtualization overhead of the software service based on the ANSYS Fluent platform. Such evaluations are crucial to guarantee an efficient overall system usage and enable other domains to execute their applications also on such large-scale systems.

One such application is the calculation of the Levenshtein distance between two strings like the DNA, which was known as a sequential-only application. To make use of the ecosystem presented before, it requires a scalable parallel implementation.

Sadiq and Yousaf [6] introduce a distributed, parallel algorithm to calculate the Levenshtein distance.

Combining all these software and middleware oriented approaches, it is mandatory to consider the actual hardware resources available. Berberich et al. [1] provide an overview on the European HPC landscape and introduce a pan-European HPC portal collecting all information and facilitate access to the portfolio of services offered across Europe.

The collection of papers presented in this special issue provides some valuable solutions for using the future exascale supercomputers effectively and efficiently. However, at the same time, many aspects are useful on smaller systems as well and demonstrate the utilization of today's technology, such as for example containers, for applications in science and research. It is clear that the exascale threshold is a merely artificial construct. The true challenge stems from the ever increasing computational performance, and the necessity of scaling applications efficiently. Under this perspective, the tools and applications presented here demonstrate what is possible with today's technology, and even more, and the mindset needed to utilize the supercomputing systems tomorrow.

Acknowledgment

We would like to thank Dr. Ladislav Hluchy, the Editor-in-Chief of Computing and Informatics (CAI) for his timely advice on this special issue. A big thanks also goes to Viera Jablonska, the CAI journal editorial assistant for her great support in publication of the special issue. This work has only be possible through the support of the European Commission within the EU project PROCESS, under the grant agreement No. 777533.

REFERENCES

- [1] BERBERICH, F.—LIEBMANN, J.—TEODOR, V.—NOMINÉ, J.-P.—PINEDA, O.—SEGRS, P.: European HPC Landscape. *Computing and Informatics*, Vol. 39, 2020, No. 4, pp. 622–643, doi: 10.31577/cai_2020.4.622.
- [2] BOBÁK, M.—HLUCHÝ, L.—HABALA, O.—TRAN, V.—CUSHING, R.—VALKERING, O.—BELLOUM, A.—GRAZIANI, M.—MÜLLER, H.—MADOUGOU, S.—MAASSEN, J.: Reference Exascale Architecture (Extended Version). *Computing and Informatics*, Vol. 39, 2020, No. 4, pp. 644–677, doi: 10.31577/cai_2020.4.644.
- [3] BARANOWSKI, M.—BELLOUM, A.—CUSHING, R.—VALKERING, O.: *Cookery: A Framework for Creating Data Processing Pipeline Using Online Services*. *Computing and Informatics*, Vol. 39, 2020, No. 4, pp. 678–694, doi: 10.31577/cai_2020.4.678.
- [4] BYSTROV, O.—KAČENIAUSKAS, A.—PACEVIČ, R.—STARIKOVIČIUS, V.—MAKNICKAS, A.—STUPAK, E.—IGUMENOV, A.: Performance Evaluation of Parallel

- Haemodynamic Computations on Heterogeneous Clouds. *Computing and Informatics*, Vol. 39, 2020, No. 4, pp. 695–723, doi: 10.31577/cai_2020.4_695.
- [5] CUSHING, R.—VALKERING, O.—BELLOUM, A.—SOULEY, M.—BOBAK, M.—HABALA, O.—TRAN, V.—GRAZIANI, M.—MÜLLER, H.: Process Data Infrastructure and Data Services. *Computing and Informatics*, Vol. 39, 2020, No. 4, pp. 724–756, doi: 10.31577/cai_2020.4_724.
 - [6] SADIQ, M. U.—YOUSAF, M. M.: Distributed Algorithm for Parallel Edit Distance Computation. *Computing and Informatics*, Vol. 39, 2020, No. 4, pp. 757–779, doi: 10.31577/cai_2020.4_757.
 - [7] GRAZIANI, M.—EGGEL, I.—DELIGAND, F.—BOBÁK, M.—ANDREARCZYK, V.—MÜLLER, H.: Breast Histopathology with High-Performance Computing and Deep Learning. *Computing and Informatics*, Vol. 39, 2020, No. 4, pp. 780–807, doi: 10.31577/cai_2020.4_780.
 - [8] GSCHWANDTNER, P.—JORDAN, H.—THOMAN, P.—FAHRINGER, T.: AllScale API. *Computing and Informatics*, Vol. 39, 2020, No. 4, pp. 808–837, doi: 10.31577/cai_2020.4_808.
 - [9] MADOUGOU, S.—SPREEUW, H.—MAASSEN, J.: Processing Radio Astronomical Data Using the PROCESS Software Ecosystem. *Computing and Informatics*, Vol. 39, 2020, No. 4, pp. 838–859, doi: 10.31577/cai_2020.4_838.
 - [10] MEIZNER, J.—NOWAKOWSKI, P.—KAPALA, J.—WOJTOWICZ, P.—BUBAK, M.—TRAN, V.—BOBÁK, M.—HÖB, M.: Towards Exascale Computing Architecture and Its Prototype: Services and Infrastructure. *Computing and Informatics*, Vol. 39, 2020, No. 4, pp. 860–880, doi: 10.31577/cai_2020.4_860.



Dieter KRANZLMÜLLER is Chairman of the Board of Directors at Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities. In 2008 he joined the Board of Directors at LRZ and became Full Professor of computer science at the Chair for Communication Systems and System Programming at Ludwig-Maximilians-Universität Munich (LMU). His scientific focus lies in e-infrastructures, including network and IT management, grid and cloud computing, as well as in high performance computing, virtual reality and visualisation. He graduated from the Johannes Kepler University Linz, Austria.

Having spent a number of years working in the IT industry, he returned to academia to work at the universities of Reading and TU Dresden, and to act as deputy director of the EGEE project at CERN in Geneva. He is truly internationally oriented and is a member of many European and international organizations in the field of IT.



Maximilian HÖB is Associate Scientist in the Munich Network Management Team at Ludwig-Maximilians-University Munich and Co-Coordinator of the PROCESS project, in which he also contributes to two Use Cases in the area of data management and agricultural simulation based on the Copernicus data sets. His research focuses on large scale system architectures and performance-aware containerization of HPC applications.

EUROPEAN HPC LANDSCAPE

Florian BERBERICH

PRACE aisbl and Jülich Supercomputing Center
Forschungszentrum Jülich GmbH
52428 Jülich, Germany
e-mail: f.berberich@fz-juelich.de

Janina LIEBMANN, Veronica TEODOR

Jülich Supercomputing Center
Forschungszentrum Jülich GmbH
52428 Jülich, Germany
e-mail: {j.liebmann, v.teodor}@fz-juelich.de

Jean-Philippe NOMINÉ

ETP4HPC and Commissariat à l'Énergie Atomique et aux Énergies Alternatives
DAM, DIF
91297 Arpajon, France
e-mail: Jean-Philippe.NOMINE@cea.fr

Oriol PINEDA

PRACE aisbl and Barcelona Supercomputing Center
Carrer de Jordi Girona, 29, 31
08034 Barcelona, Spain
e-mail: oriol.pineda@bsc.es

Philippe SEGERS

PRACE aisbl and Grand Équipement National de Calcul Intensif
6 bis rue Auguste Vitu
75015 Paris, France
e-mail: philippe.segers@genci.fr

Abstract. This paper provides an overview on the European HPC landscape supported by a survey, designed by the PRACE-5IP project, accessing more than 50 of the most influential stakeholders of HPC in Europe. It focuses at Tier-0 systems on the European level providing high-end computing and data analysis resources. The different actors are presented and their provided services are analyzed in order to identify overlaps and gaps, complementarity and opportunities for collaborations. A new pan-European HPC portal is proposed in order to get all information on one place and facilitate access to the portfolio of services offered by the European HPC communities.

Keywords: European Commission, EC, European, high performance computing, HPC, ecosystem, exascale, services, platform, EuroHPC, PRACE, ETP4HPC, CoE

Mathematics Subject Classification 2010: 68-00

1 INTRODUCTION

The European Commission (EC) recognised the need for an EU-level policy in High-Performance Computing (HPC) to optimise the national and European investments in the field and to coordinate the entire HPC ecosystem. To this end, on 15 February 2012 the EC strategy on HPC was published in the communication “High Performance Computing: Europe’s place in a Global Race” [1]. Acknowledging the importance of HPC for society, science and industry the communication announced a joint European effort in order to increase the investments in HPC and promoting European HPC technology. This strategy is taking shape, with the creation of the EuroHPC Joint Undertaking (JU) [2].

As no single member state alone has the financial and human resources to develop a sustainable exascale¹ HPC ecosystem, within the European Data Initiative (EDI) subpart of the Digital Single Market (DSM) strategy [3], the EC has step by step increased the investment in HPC significantly [4], supporting with various projects and initiatives the three HPC pillars, as shown in Figure 1: Technologies, Infrastructure and Applications with a strong pan-European coordination. Moreover, a strong HPC ecosystem has been identified as mandatory to leverage the full potential of data in Europe, along with the need for providing more open data, coping with interoperability issues and fragmentation of access to data and digital services.

Back in time, first computers were supercomputers, by definition, the most powerful systems of their time, when computers were huge machines, dedicated to solve a specific industrial or academic problem that could not be addressed without them,

¹ Exascale: HPC systems at the scale of 10^{18} floating-point operation per second. US, Japan and China investments for exascale are of the order of one billion € each, to acquire these systems near 2020.

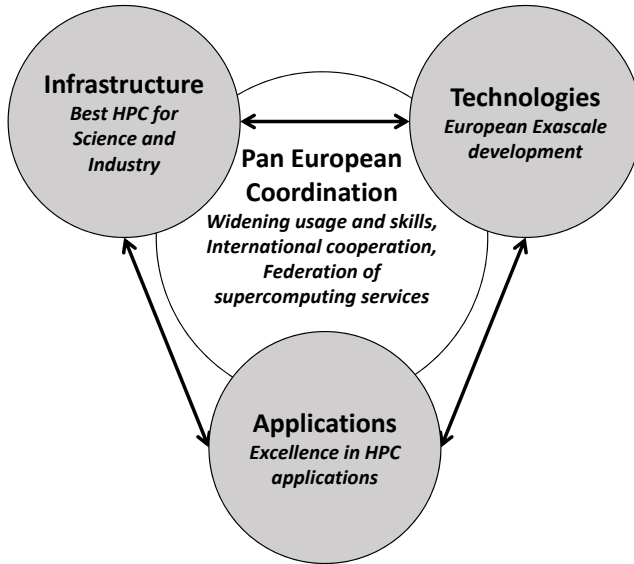


Figure 1. HPC ecosystem pillars

from the first Turing machine to the systems used for the Manhattan project [7] (Oak Ridge National Laboratory [9], one of the leading pre-exascale computing centre being a direct spin-off of this early development of supercomputing). In these early years, the three pillars worked together in each single project, designing the technology and the infrastructure to support it at the same time. The same people worked at the application to solve their problem with the system designed for that. It is only in a second phase that the industry diverged, with one branch that kept addressing the biggest problems of its time, on dedicated systems with specific technologies, a niche but also a strategic market. The second branch, mainstream, evolved to solve problems for a wider market, from industry to office applications, and eventually became the mass market that it is now.

With the rise of the information technology industry, computers becoming a huge mass market, HPC industry partially moved from specific technologies, especially on processor, to general purpose processors, when the number of parallel cores increased and it became more efficient to assemble many general purpose processors than fewer specific ones. Of course, other parts of a supercomputer still needed specific components, with higher performance than mass market components, such as the interconnect, because a supercomputer is much more than just many processors. But from that point on, HPC was not anymore the biggest driver of processor and chips technology, even if the processors used for supercomputers were still the high-end of technology providers, the portfolio of products was not anymore designed for them.

Nowadays, there is a clear distinction between Infrastructure, Application and Technology providers. But for the last miles of the exascale run, with the small leverage HPC could have on the whole information technology market, the co-design is more important than ever. A coordination is needed, especially at the European level missing the same intrinsic coordination as seen in the USA, China, Japan or other key players in the exascale league. Some specific coordination, research and innovation actions are needed at the European level to articulate the interaction of these actors in the most efficient way. Coordination will be also of the upper importance for future quantum computing technologies, which will be closely related to HPC but who will need the development of a new paradigm of the algorithm to solve problems in a way that takes advantage of the specificity of quantum theory, with the strength and weakness of their transposition to the computing industry. Efficiency will ultimately derive from the coordination of these three pillars. This article analyses the current situation, presents the three pillars of the HPC ecosystem, their services, with their overlaps and gaps, complementarity and opportunity for collaborations, and proposes a high-level service architecture.

2 THREE PILLARS OF THE HPC ECOSYSTEM

The European HPC Strategy is based on three pillars: Technologies, Infrastructure and Applications. Each organisation, project or initiative is linked to at least one of those pillars. These three pillars have the objective to serve the European HPC user community through a user-driven approach, with the relevant European HPC communities and user groups adequately represented in each of the pillars. The individual actors are described briefly in this section.

EuroHPC Joint Undertaking (JU) [2] was founded on 28 October 2018. EuroHPC JU will permit the EU and participating countries to coordinate their efforts and share resources with the objective of deploying in Europe a world-class supercomputing infrastructure and a competitive innovation ecosystem in supercomputing technologies, applications and skills. A good overview on the European exascale projects, (FETHPC and Centre of Excellence (CoE)) is provided by the European High-Performance Computing Handbook 2018 with an update 2019 [6].

This schematic view of a complex ecosystem of stakeholders should not be seen as something static, with homogeneous pillars and some formally established relation between them, but more like a dynamic “n-body” interactions, with constantly evolving needs and offers of these pillars. Actors are constantly interacting between these pillars, with some of them playing active roles on more than one pillar. To provide one short example, Technologies are evaluated through benchmarks of Applications, representative of typical workload of HPC Infrastructure, and co-design is part of the HPC culture with overlaps and complementarity among the pillars.

In addition to the three classical pillars also the convergence of simulation and big data workloads – due to the deluge of data coming from next generation sci-

tific instruments (satellites, (radio)telescopes, accelerators, microscopes, sequencers, etc.) – are becoming more and more important. The same development is being monitored in the case of Internet of Things (IoT), social media and large scale simulations (massive 3D simulations, multi-scale and multi-physics coupled simulations, ensemble/optimisation/scenario studies, uncertainties quantification, etc.).

A HPC infrastructure tailored to treat these large amount of data will be also indispensable for machine learning (ML) or artificial intelligence (AI). Data are involved in all three pillars. The Technologies pillar will develop systems designed especially for I/O or AI, the Infrastructure pillar will provide access to systems suitable for AI, and in the Applications pillar ML and AI algorithms will be implemented in codes [19, 26, 28, 30, 31]. Quantum computing, still at an early stage of industrial development, could also be represented by such three pillars, with promising development of technologies providing prototypes of bigger scale at a rapid pace. The work on the application side to reframe problems in a way that can be handled by quantum system is a very new field, in addition to the simulation of future quantum computer (with their specific “speed” of calculation but also their specific “error rate” related to the statistical nature of a quantum computation result). And lastly, the usefulness of such new technologies will be highly dependent on their implementation into existing or new infrastructure to be able to provide access to end-users.

2.1 Infrastructure

2.1.1 Partnership for Advanced Computing in Europe (PRACE)

The development of the European HPC ecosystem was initiated and pursued by the Partnership for Advanced Computing in Europe (PRACE) and its 26 partners in the past ten years. PRACE is supported by the PRACE member states and through the EU by a series of implementation phase (IP) projects [24]. Over the last decade, PRACE and its partners have given national HPC ecosystems a common European umbrella which is recognised as an ESFRI Landmark since 2016. PRACE was founded in 2010 as the European HPC infrastructure, with an investment above 400 million Euros from four hosting members for its first phase (ES, DE, FR, IT), with the objective of developing a persistent and pan-European HPC facility. In the second funding period of PRACE five European countries (with the addition of CH) committed to host leading-edge supercomputers on the highest performance level in Europe. Subsequent investments from all PRACE members (from 26 countries) and contributions from the EC have allowed the infrastructure to provide a continuous set of services, based on a peer-review process assessment of scientific excellence, awarding more than 25 billion core hours more than 700 research projects led by investigators from 40 different countries, from academia and industry. This large computing capacity has been complemented by high-level quality training and strong user support programmes (including support to SMEs), in order to foster the development of a solid HPC community in Europe.

PRACE, with a governance structure that includes the Scientific Steering Committee (SSC) and the Industrial Advisory Committee (IAC), underlined the urgent need for more compute cycles, and huge demands in terms of memory/storage capacities and performance in the recent Scientific Case 2018-2026 [11]. Also the need for new approaches, i.e. scaling via ensembles, deep learning, and statistical models are expressed.

In relation to EuroHPC JU, PRACE published its Position Paper: PRACE in the EuroHPC Era [10] which defines its current and proposed future services for the European HPC ecosystem. The following services encompass the PRACE offer:

1. Peer Review Access to HPC systems,
2. Support for industry (including SMEs),
3. Enabling of HPC applications,
4. Services for universities and user communities,
5. HPC training,
6. Promotion of HPC careers-gender balance,
7. Operational HPC services,
8. HPC procurement and prototyping support,
9. Dissemination and documentation for HPC services.

Many of the PRACE services (1–5) are also reflected in the different fields (see Figure 2) we defined for classification of the European HPC Landscape:

1. HPC Policy,
2. HPC Technology,
3. HPC Computing Services,
4. HPC Training,
5. HPC Application Enabling and User Support,
6. HPC Research.

Only HPC policy and HPC technology are not directly included in the PRACE position paper, because PRACE is not providing HPC policy and direct HPC technology, even if it provides some evaluation of technologies and guidance through a series of white papers and best practice guides [22, 23], illustrating the strong interaction already in place between those pillars.

2.1.2 GÉANT

GÉANT develops, delivers and promotes advanced network and associated e-infrastructure services for research and education, supporting open collaboration and knowledge-sharing amongst its members and the wider research and education community. The GÉANT Association BV [8] is owned by its core membership of the

European National Research and Education Network (NREN) organisations. Since coordinating pan-European research and education (R & E) networking on behalf of Europe's NRENs the GÉANT's role has evolved to that of a true services innovator, incorporating network planning, procurement, building and operation, as well as coordination of research programmes and development of innovative services. Working with NREN partners and the EC, the high-speed networks that they build and operate connect NRENs to each other and to the rest of the world, enabling scientists, academics, innovators and students to collaborate, regardless of their location. HPC current and future new exascale usages highly depend on the integration of core HPC services with data and network services.

2.2 Applications

The importance of a strong HPC applications ecosystem has been periodically highlighted by the PRACE SSC [11] and PRACE User Forum. This has been acknowledged by the EC [1] through the large funding allocated to this pillar. Applications are the core intellectual properties of many communities, a strong asset for European research, both in academic and industrial fields where Europe has often acquired a leading position worldwide. To coordinate the needed effort, Centres of Excellence in computing applications (CoEs) have been designed to address specific needs of communities (weather and climate, material science, medicine, etc.) or transverse needs (industry, algorithm, etc.). After a first selection of CoEs in 2015, the European Commission reviewed the communities supported by awarding a second generation of nine CoEs in 2018 [13], then an additional extra group of four in 2019 [32]. These cover a wide collection of scientific domains, including bioinformatics, biomedical sciences, climate sciences, energy and engineering, materials, social sciences and solid earth, as well as the transversal topic of computing performance, each of them covered by one CoE. The second generation of CoEs builds on the success of the first selection, and will highly contribute to strengthen Europe's leadership in HPC applications through their associated services, such as: developing, optimising (if needed re-designing) and scaling HPC application codes towards peta- and exascale computing; testing, validating and maintaining codes and managing the associated data; quality assurance; co-design of hardware, software and codes; consultancy to industry and SMEs; research in HPC applications; and addressing the skills gap in computational science. The FocusCoE project [33] supports the CoEs to more effectively fulfil their role within the ecosystem and ensure that extreme scale applications result in tangible benefits for addressing scientific, industrial or societal challenges.

This strong contribution to the applications pillar is further enhanced by the highly specific application developments of FETHPC projects and EPI (European Processor Initiative) [8], and with code-enabling activities of PRACE-IP projects and PRACE High Level Application Support Teams (HLSTs) [25]. Beside of that, new usage domains for HPC are developing, such as Humanities or Artificial In-

telligence, with different needs and constraints, requiring innovative ways to access resources.

2.3 Technology

The ETP4HPC Association [14] was created in 2012, to be the voice of the HPC suppliers and promote HPC technologies development, and in particular to prepare input and R&D recommendations to the EC in this area. In 2014 a “contractual Public Private Partnership” in HPC (cPPP) was signed between the EC and ETP4HPC association [13]. A significant fraction of the funding provisioned under this cPPP was assigned to a series of calls on HPC technology R&D [14]: the so-called FETHPC calls (part of the “Future and Emerging Technologies” branch of the successive H2020 Work Programmes). Between 2014 and 2018, 32 FETHPC projects were selected with a total funding of approximately 175 M€. Meant to develop HPC systems hardware and software building blocks in the areas of HPC node architecture, system and middleware, programming environment and tools, the FETHPC projects already produced a number of innovations and prototypes [15, 16], co-developed between technology suppliers (large companies or SMEs), research organisations and end users, sometimes leveraging other innovations dedicated to the wider market of data centres as a whole.

The ETP4HPC Strategic Research Agenda [13], updated every 2 years since 2013, has been the main source of advice and influence regarding the FETHPC calls contents. ETP4HPC also actively participates in the overall EU HPC ecosystem development. ETP4HPC members are technology suppliers and research organisations.

In addition, in 2017–2018 the EC also implemented an important new call to establish a Framework Partnership Agreement on European low-power microprocessor technologies to establish a stable and structured partnership between the EC and committed institutions and organisations. The EPI consortium was selected to co-design, develop and bring to the market a European low-power microprocessor, one of the core elements needed for the development of the European supercomputers with exascale capacity [18, 8]. The co-design aspect of EPI is a key factor to provide a next generation of processors that fully harvest the benefits of energy efficiency for relevant European applications.

Two years after its establishment, the EuroHPC JU [2] is now being implemented and ramping up. EuroHPC JU is taking over from the HPC cPPP to continue the HPC R&D funding towards exascale, from 2019 onward, more strongly coordinating the follow-ups of FETHPC and EPI projects. Members of the EuroHPC JU are the EC, 32 EU member and associated states, and the private members ETP4HPC and the BDVA (Big Data Value Association [30]). ETP4HPC is represented in the EuroHPC Research and Innovation Advisory Group (RIAG).

The outcomes of the cPPP phase – 2014–2018 – have been documented by the annual Progress Monitoring Reports (PMRs [27]). The 2018 PMR (published in the end of summer 2018) summarises this 4-year period of joint support of HPC

technologies and applications by H2020. In particular positive effects are observed regarding job creation (both in research and HPC supply industry in Europe), intellectual property creation, and private companies extra investments – which leverage the public funding effort in initial R & D in order to productise solutions and bring them to the market. A number of European SMEs in particular have been clearly benefitting from H2020 funding and correlatively augmented their staff, business and turnover.

Since the EC funded CoEs as well as FETHPC projects, it also supported the evolution and improvement of many HPC applications, in addition to many innovative hardware and software building blocks for HPC solutions. This helped CoEs contribute to evolutions of community codes (in terms of features and/or portability and/or performance improvement and scaling).

These efforts are now smoothly continued in the EuroHPC Research & Innovation Pillar from 2019 onward.

2.4 Pan-European Coordination

EuroHPC JU already set up two R & I work plans, for 2019 and 2020, the first two years of EuroHPC functioning in the context of Horizon 2020 framework programme. We describe the related calls for projects below. Since EuroHPC R & I Pillar encompasses all aspects not related to Infrastructure procurements and operations, we can find in these work plans a mix of calls that are related to either Technologies or Applications ecosystem pillars: The EuroHPC JU Workplan 2019 [34] encompassed two sets of calls for proposals:

1. Towards Extreme Scale Technologies and Applications which has two facets (a technologies call with support for hardware and software building blocks, and two calls relating to applications).
2. Innovating and Widening the HPC use and skills base has two calls on the users and skills we relate to the Applications pillar.

Regarding the “Towards Extreme Scale Technologies and Applications” call [35]:

- Nine selected projects addressing the topic 1, extreme scale computing and data driven technologies, are expected to address performance and efficiency of future exascale systems.
- Five selected proposals address the call topic 2, HPC and data centric environments and application platforms, and will focus on the development of energy-efficient HPC software. The projects are expected to demonstrate significant use cases and pilot systems.
- Five selected proposals address the call topic 3, industrial software codes for extreme scale computing environments and applications, and are expected to further develop, adapt and optimise HPC software for applications in the European industry.

With funding from the EuroHPC JU, EuroCC and CASTIEL projects will build a European network of 33 national HPC competence centres. The two projects will bridge the existing HPC skills gaps while promoting cooperation and the implementation of best practices across Europe. Each of the 33 national competence centres, which will be part of the EuroCC network, will act locally to map available HPC competencies and identify existing knowledge gaps [38].

The EuroHPC JU work plan 2020 has three main dimensions:

1. Two so-called pilot calls which are clearly related to Technologies pillar, with a system-wide vision, typically meant to integrate building blocks (previously developed or developed in the course of the new projects).
2. An EPI follow-up call, clearly a Technologies pillar aspect, compute hardware oriented.
3. A future call on Education and Training, on the side of Applications pillar addressing widening usage and skills.

EuroHPC work plan 2020 then planned different calls, two of them now being closed [37]:

1. Advanced pilots towards the European supercomputers,
2. A pilot on quantum simulator.

More generally, EuroHPC private members (ETP4HPC and BDVA) are sustaining the development and updates of their research agendas, together with the HPC wider ecosystem and leading stakeholders and representative entities such as PRACE, CoE representatives, AIOTI [28], and also with international collaborations (such as BDEC [30]). Taking into account not only Big Data but also AI and IoT trends in advanced computing is a necessity. The point is to develop HPC both towards extreme scale (exascale and beyond) and also to extend its use and insert it in a digital continuum from edge to cloud and bigger centralised but interconnected HPC centres. ETP4HPC and BDVA help the communities express recommendations and priorities towards EuroHPC Advisory Groups, and it is Governing Board which eventually decides on R&I funding.

3 ANALYSIS OF THE HPC LANDSCAPE

3.1 Summit

An HPC Ecosystem Summit was organised by PRACE on 14 May 2019 during the EuroHPC Summit Week 2019 in Poznan, Poland. The summit was attended by more than 50 representatives from the European Commission, PRACE, GÉANT, CoE and FETHPC projects, EXDCI and ETP4HPC, among others. The objective of this summit was to present current activities and discuss future roles and responsibilities of the key European HPC stakeholders within the landscape. The outcome of this

summit was expected to furnish a vision of the architecture and integration of the HPC services with European Open Science Cloud (EOSC) [29], EDI, data services, etc. for the communities. The discussion during this summit allowed to clarify the results of a preparatory survey and to define further stratification actions.

3.2 Survey

To prepare for the HPC Ecosystem Summit, a dedicated survey with 10 questions was sent to 81 contacts (coordinators of CoEs, FETHPC projects, EuroHPC JU, ETP4HPC, BDVA, GÉANT, FocusCoE, EPI, EOSC, EUDAT, OpenAire, eInfra-Central, EXDCI). For bigger projects or umbrella organisations only the coordinator of the project or organisation was contacted. The contacts were asked in more detail to indicate which of the three pillars they are part of, if they would be able to attend the HPC Ecosystem Summit in order to take part of the discussions, to indicate their specific domain and include a list of their services. Moreover, they were asked to indicate possible overlap and collaboration with other initiatives or organisations. The main part of the survey was the self-evaluation of the current actors concerning their role in the European HPC landscape provided through the answers to the following matrix, as shown in Figure 2.

	Developer	Coordinator	Provider	User/Beneficiary	Enabler
HPC Policy					
HPC Technology (industry, hard & soft)	<ul style="list-style-type: none">• <i>Developer: institution in charge of preparing materials for the development activity</i>• <i>Coordinator: institution in charge of collecting materials from developers and of coordinating their implementation</i>• <i>Provider: institution in charge of providing the services to execute the activity</i>• <i>User/beneficiary: institution that benefits from the activity</i>• <i>Enabler: institution that enables the activity by providing the necessary services that are not part of the core of the activity</i>				
HPC Computing Services					
HPC Training					
HPC Application Enabling and User Support					
HPC Research					

Figure 2. Self evaluation matrix of the role in the European HPC landscape

3.3 Services

3.3.1 HPC Policy

EuroHPC JU is developing the policy in terms of funding and the main guidelines. The Research and Innovation Advisory Group (RIAG) and the Infrastructure Advisory Group (INFRAG) are the information gathering bodies in the EuroHPC JU. This will include inputs from PRACE, ETP4HPC and from CoE and FETHPC projects, as shown in Figure 3 a).

3.3.2 HPC Technology

The provision of HPC technology should be driven by the FETHPC projects and the European Processor Initiative (EPI) with the goal to develop European technology for exascale computing. Surprisingly, only some FETHPC projects see themselves as HPC technology provider. However, 80 % of the FETHPC projects declared to develop HPC technology. Some CoEs also indicated a contribution to the HPC technology, since there are some of them with a co-design approach, see Figure 3 b). A detailed analysis has been done by the EXDCI-2 project [5].

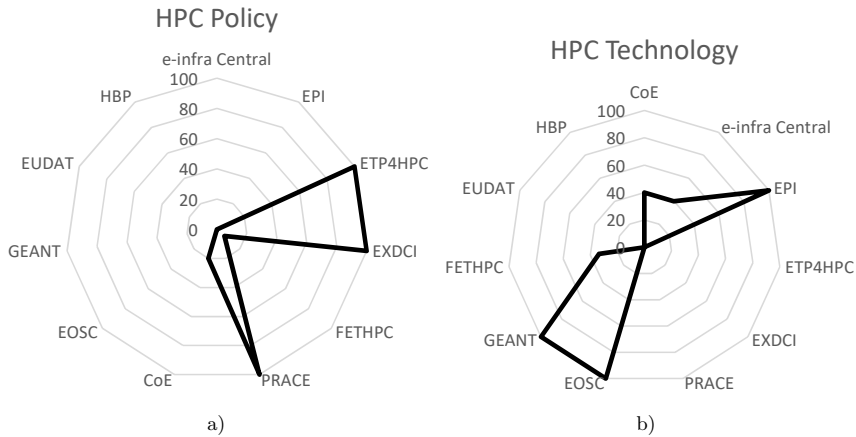


Figure 3. Provision of a) HPC Policy and b) HPC Technology in % of positive responses in the respective stakeholder groups

3.3.3 HPC Computing Service

The provision of HPC Computing Service refers to making available HPC resources for testing, scaling and production. PRACE via its members is the major European HPC resources provider, as shown in Figure 4 a). Additionally, EOSC and Human Brain Project (HBP) also presented themselves as resource providers. Indeed, EOSC provides access to existing services that are compliant with EOSC rules, though at a lower scale compared to PRACE resources. The FENIX [40] federated set of e-infrastructure services also provides access to HPC resources infrastructure via the ICEI project (Interactive Computing E-Infrastructure for the Human Brain Project – HBP), funded by the EC in the context of the Framework Partnership Agreement of the flagship project HBP. The distinguishing characteristic of this e-infrastructure is that data repositories and scalable supercomputing systems are in close proximity and well integrated, providing a generic e-infrastructure for HPC, driven by its scientific use-cases and usable by other scientific communities, such as the ones from EOSC, but also future ones such as the one from the Square Kilometer

Area telescope project SKA [42]. This is also a good example of coordinated action, using state of the art HPC, storage and network technologies to build an application layer that helps user to access a portfolio of HPC services in a convenient way.

3.3.4 HPC Training

The results from the survey showed a significant number of contributors to HPC Training services, including PRACE, HBP, EOSC, CoE and FETHPC projects. In Figure 4b) the percentage of the received positive answers is shown. While the trainings from HBP, EOSC and FETHPC were identified as independent and complementary, a potential overlap was identified between the training offer of PRACE and that of the CoEs. This had been already identified in previous discussions and through the FocusCoE coordination action, where a decision was taken to focus PRACE training on general and cross-disciplinary HPC topics, while CoEs would focus on topical trainings.

In order to make all training offers well-known, they are collected and will be made available in a centralised European portal [41]. This will be based on a joint training database, to be fed with training offers from all national and European actors; this database will be shared and will include categories to allow searching for specific trainings.

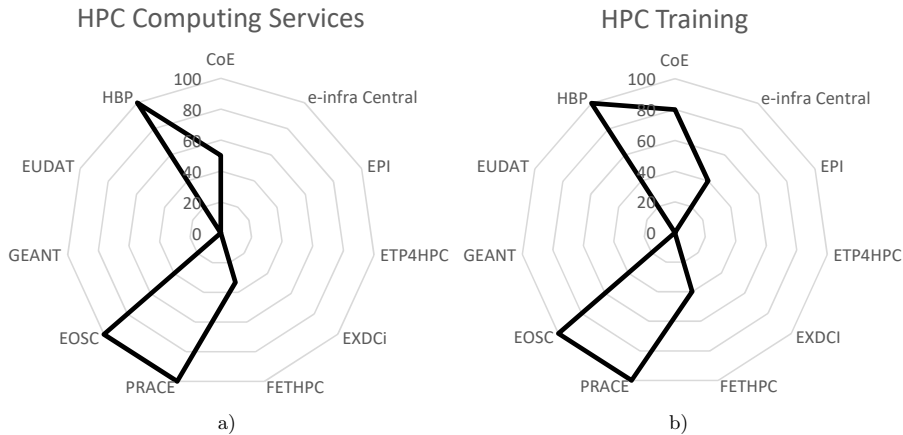


Figure 4. Provision of a) HPC Computing Service and b) HPC Training in % of positive responses in the respective stakeholder groups

3.3.5 HPC Application Enabling and User Support

The survey showed again a significant number of actors contributing to HPC Application Enabling and User Support activities, as presented in Figure 5 a). After the

discussion, it was concluded that this item required further stratification according to the additional dimensions of support levels and targeted users.

HPC support is classified in four levels depending on the scope of the support provided, which ranges from short helpdesk support to long-term refactorization support, including horizontal performance analysis by the POP CoE. While some overlap could initially be identified in medium-term support (level 2 and level 3) between the PRACE HLST [25] programme and CoEs, this was discriminated through their target users.

Similarly to training, the available support catalogue will be collected and made available in a centralised EU portal managed by PRACE. Further analysis on this service will be carried out by the PRACE-6IP project.

3.3.6 HPC Research

The survey showed that research in HPC is mainly executed by the actors in the HPC pillar of applications, that is CoE and FETHPC projects. This would include also EPI when one considers research in HPC technology. However EPI indicated HPC Research as Developer and not as Provider.

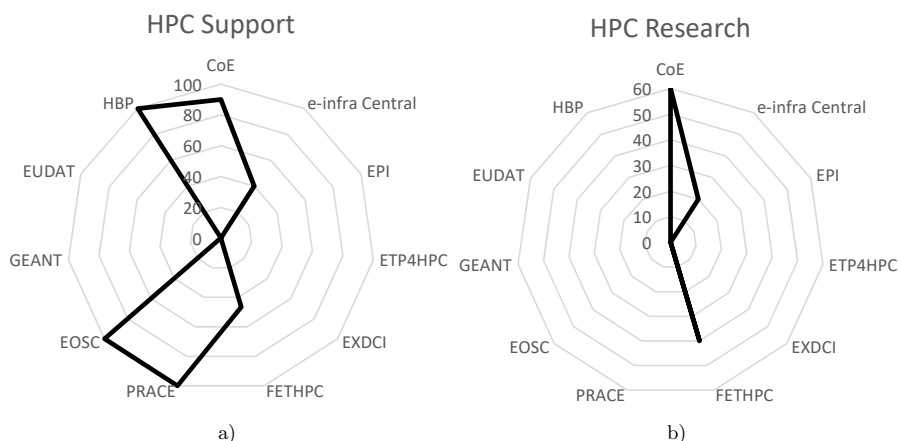


Figure 5. Provision of a) HPC User Support and b) HPC Research in % of positive responses in the respective stakeholder groups

3.4 Link to Other e-Infrastructures

PRACE is preparing for the use of HPC and data resources by other infrastructures and e-infrastructures. With the increasing amount of data traditional workflows will have to be changed in order to cope with the data. This is especially true for large scale scientific instruments, e.g. CERN or the SKA telescope. In order

to tackle this aspect, CERN, SKAO, GÉANT and PRACE formed a pioneering collaboration (officially signed in July 2020 [39]) that works to help realise the full potential of the coming new generation of HPC technology. During the initial period of 18 months, the collaboration is developing a benchmarking test suite and a series of common pilot “demonstrator” systems such as training, authenticated workflows, benchmarking and data access. The activity of these demonstrators has been kicked-off in September 2020 with a joint meeting of all the four leading research organisations.

The European Open Science Cloud (EOSC) has been designed to increase the value of scientific data assets by making them easily available to a greater number of researchers, across disciplines (interdisciplinarity) and borders (EU added value), and to reduce the costs of scientific data management, while ensuring adequate protection of information/personal data according to applicable EU rules. EOSC is one of the major actions of the Communication on a “European Cloud Initiative” of April 2016 [3]. The European Open Science Cloud will offer 1.7 million European researchers and 70 million professionals in science and technology a virtual environment with open and seamless services for storage, management, analysis and re-use of research data, across borders and scientific disciplines by federating existing scientific data infrastructures, today scattered across disciplines and member states. Part of EOSC’s mission is to join the existing and emerging data infrastructures. To be part of EOSC, the infrastructure should comply with the rules of participation that define the rights, obligations and accountability of its various actors.

As one of its underlying layers, PRACE is not formally part of the EOSC but is compliant with main rules of participation, working on a referenced joint service catalogue, enabling users to find and access the HPC resources also in the EOSC. While new complementary ways to access converged HPC and AI resources in Europe are foreseen (such as AI4EU [19]) to respond to new usage models, the allocation of large resources in HPC and also in other fields of science will probably still be based on a peer-review process assessing scientific excellence, which is not contradictory with EOSC rules. In addition, it is planned also to integrate EOSC training in the HPC in Europe portal (see the next section) and make the training offers accessible also for EOSC and HPC ecosystem.

4 NEXT STAGE – HPC PORTAL

In order to facilitate the access to HPC resources and other linked services, the conclusions from the analysis of the HPC landscape have been used to shape a new European HPC services portal. This portal will serve as a central HPC services database gathering the offers and services of all European HPC actors. The new HPC in Europe portal will classify the elements collected according to three different criteria:

- Service nature: HPC Access, Training and Events, Support, Applications, Tech-

nology and Documentation.

- Target: Research, Industry (with a dedicated section for SMEs), Skills development, HPC Communities and General Public (including stakeholders and policy makers).
- Maturity level: from basic/beginner to advanced/experienced.

This structure will facilitate the navigation through the entire European HPC services catalogue and provide a clear access to all of them. The objective of the portal is to collect and display the first level of information, that is, a short but clear description of each service and the most relevant constraints (e.g.: dates, eligibility), in order to guide HPC users to the services that meet their needs. The portal will also include import and fetching capabilities, in order to feed and fetch the services from all other HPC European providers. The descriptors and categories of each service will be agreed with the major stakeholders.

The HPC in Europe portal [41] has been designed to be complementary to the classification of HPC services and related activities according to different target audiences (i.e. researchers, students, industry and projects) available in the EXDCI portal [21]. The difference between both portals is the scope of the services and the level of information provided. Both portals might be merged in the future. The details and current status of the portal strategic development is summarised herein:

- The elements within the different service natures have been further categorised based on the feedback received from the HPC stakeholders during the workshops organised to this effect. The category tree is being tested with real services provided by these same stakeholders.
- A number of highlights have been selected to appear on the front page of the portal to give a direct access to the related services.
- A dedicated Training section will offer a joint catalogue of HPC trainings provided by all training providers in Europe including PRACE, CoEs, EOSC, FETHPC projects and HBP, and also national training offers; this will be further enhanced with training materials for different user levels and links to other related training portals worldwide.
- Likewise, all European HPC Application Enabling and User Support services will also be collected and displayed in a dedicated section, including different categories for support levels and target users.
- An interactive map will display in a graphical manner all European HPC systems, CoEs and Competence Centres and training opportunities. This will allow the users of the portal to find easily the resources and competences needed based on their location across Europe.
- The portal will incorporate a light and independent branding, not directly related to any of the HPC stakeholders. All contributors will be equally acknowledged in a dedicated section. These two measures will illustrate the neutral position of this portal towards the European HPC ecosystem.

- FocusCoE and CASTIEL recently joined PRACE in making all their services visible on the HPC in Europe portal while still maintaining the projects identities.

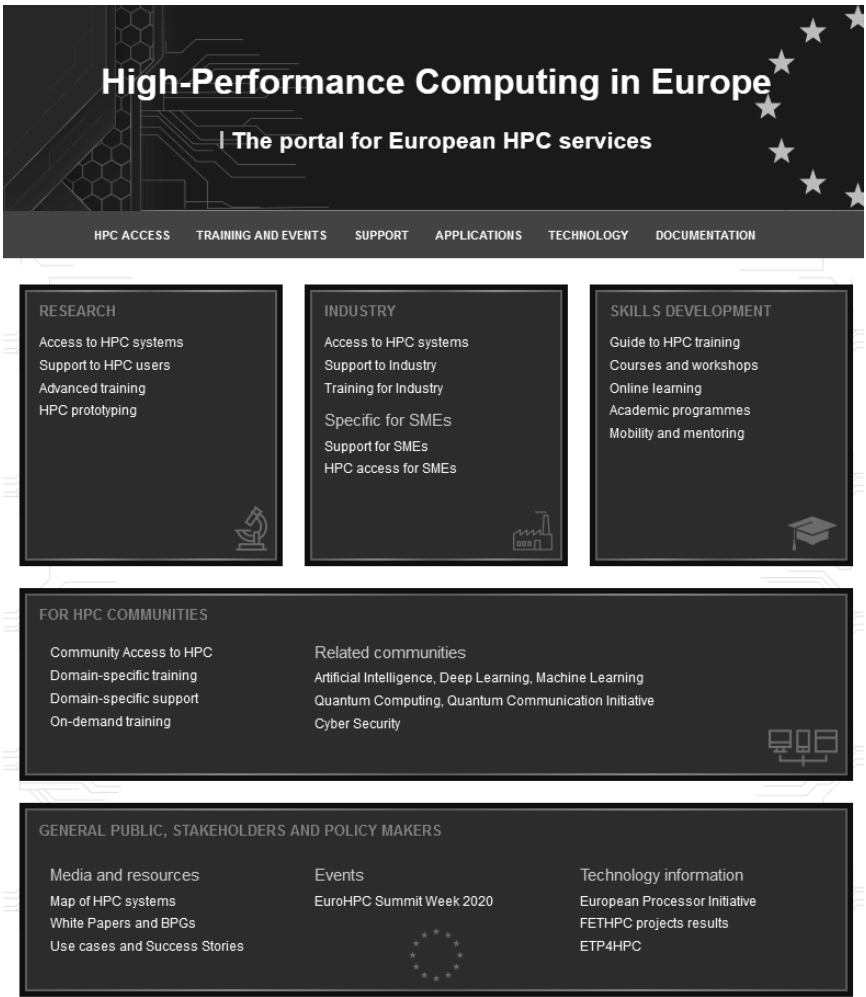


Figure 6. HPC portal

The portal will be complemented with information related to the three European HPC pillars of Infrastructure, Applications and Technologies, and further enhanced with related use cases and other HPC documentation.

The final structure of the portal includes 33 categories to further stratify the services in the six natures defined, as follows:

- HPC access: Access to HPC systems, Cloud computing, Complementary HPC access, Development and benchmark access, Prototype HPC Access and European HPC systems.
- Training and events: Training events, HPC events, On-demand training offers, Online training, Training materials, Academic programmes, Mobility and mentoring.
- Support: High-level HPC support, Domain-specific support, Support to Industry.
- Applications: European Centres of Excellence in HPC, Simulation services, Consulting services, Prototyping services, Benchmarking services.
- Technology: Software developments, Hardware developments, HPC prototyping.
- Documentation: Map of HPC systems, White Papers and BPGs, Use cases and Success Stories, HPC Media, ETP4HPC, European Processor Initiative, FETHPC projects results, Artificial Intelligence, Deep Learning, Machine Learning, Quantum Computing, Quantum Communication Initiative, Cyber Security.

A test version of the portal, including the work design is already available to facilitate the fine tuning and final improvements before its final release. Figure 6 shows this design and structure.

5 CONCLUSIONS

After the analysis of the services of the different actors in the European HPC ecosystem, it is clear that further coordination is needed at European level to leverage the strong, but scattered, skills of European players, in order to compete with united effort from USA, China or Japan on the race to exascale. To that end, the following high-level service architecture is proposed:

- European HPC technologies will be developed by the FETHPC projects and EPI. CoEs and PRACE will collaborate by providing user requirements and co-designing the new technologies. In addition, PRACE will support in providing access to the FETHPC prototypes and give the related infrastructure feedback while CoEs and other users will provide end-user feedback.
- Access to HPC and AI resources of EuroHPC JU will be provided mainly by PRACE through peer-review process for Grand-Challenge level of computation or through EOSC portal for smaller computations, while network connectivity will be provided by GÉANT. Other, more specific computing services, such as Interactive Computing, scalable computing or virtual machine and data services, along with authentication and authorisation services will be provided through a federated way using FENIX.
- The wide offer of training in HPC will be provided by many actors, where PRACE will focus on the general and cross-disciplinary training, leaving topical training to the specialised communities and CoEs.

- Application enabling will be provided by PRACE as the first contact point, along with high-level support to implementation through PRACE HLSTs. Long-term support and specific support for codes of general interest will be provided by the relevant CoEs. The POP CoE will provide transverse performance analysis services.

This high-level provision of HPC services by PRACE, GÉANT, CoEs, EPI and FETHPC projects will be complemented by the pool of national HPC services, to be integrated within this architecture. The new HPC in Europe portal will collect in this way the complete catalogue of HPC services throughout Europe, with a special emphasis in computing, training and support services. Following a user-driven approach, this will significantly increase the awareness for European HPC resources and services, and strongly facilitate their access by all European audiences. The important role of the EuroHPC JU will be acknowledged in the portal as well.

The Infrastructure Advisory Group (INFRAG) and Research and Innovation Advisory Group (RIAG), two advisory groups of the Industrial and Scientific Board of EuroHPC JU received the mission to work on the European HPC landscape. This overview and analysis will be helpful in the discussion and definition.

Acknowledgment

This work was supported by the PRACE-5IP and PRACE-6IP projects co-funded by the EUs Horizon 2020 research and innovation programme under grant agreements EINFRA-730913 and 823767.

REFERENCES

- [1] High Performance Computing: Europe's Place in a Global Race. COM(2012), 45 Final.
- [2] Council Regulation (EU) 2018/1488. Available at: <https://eurohpc-ju.europa.eu/>.
- [3] European Cloud Initiative – Building a Competitive Data and Knowledge Economy in Europe. COM(2016), 178 Final.
- [4] Proposal for a Council Regulation on Establishing the European High Performance Computing Joint Undertaking. COM(2020), 569 final, 18.9.2020.
- [5] EXDCI-2 Deliverable D2.4: Report on Coordination of the Technology Research Action in Europe. May 2020.
- [6] European High-Performance Computing Handbook 2018. ETP4HPC and EXDCI, 2018. ISBN 9789082169492. Update available at: https://www.etp4hpc.eu/pujades/files/ETP4HPC_Handbook_2019_web.pdf.
- [7] KELLY, C. C.—RHODES, R.: Manhattan Project: The Birth of the Atomic Bomb in the Words of Its Creators, Eyewitnesses, and Historians. Black Dog & Leventhal, 2009.

- [8] <https://www.european-processor-initiative.eu/>.
- [9] <https://www.ornl.gov/>.
- [10] PRACE Position Paper: PRACE in the EuroHPC Era. 2019, available at: <http://www.prace-ri.eu/about/positionpapers>.
- [11] The Scientific Case for Computing in Europe 2018–2026. PRACE Scientific Steering Committee, 2018. ISBN: 9789082169492. Available at: <https://prace-ri.eu/about/scientific-case/>.
- [12] <https://www.geant.org>.
- [13] <https://ec.europa.eu/digital-single-market/en/high-performance-computing-contractual-public-private-partnership-hpc-cppp>.
- [14] <https://www.etp4hpc.eu> and <https://www.etp4hpc.eu/cppp.html>.
- [15] <https://www.etp4hpc.eu/news/156-2017-handbook-of-european-hpc-projects.html>.
- [16] https://www.etp4hpc.eu/pujades/files/ETP4HPC_Annueal-Report-2018_web.pdf.
- [17] <https://www.etp4hpc.eu/sra.htm>.
- [18] <https://ec.europa.eu/digital-single-market/en/news/european-processor-initiative-consortium-develop-europes-microprocessors-future-supercomputers>.
- [19] <https://www.ai4eu.eu/>.
- [20] <http://www.hpc-portal.eu>.
- [21] <https://exdci.eu/about-exdci/exdci-at-a-glance>.
- [22] <https://www.prace-ri.eu/best-practice-guides/>.
- [23] <https://www.prace-ri.eu/white-papers/>.
- [24] <https://www.prace-ri.eu/public-deliverables>.
- [25] PRACE High Level Support Teams. Available at: <http://www.prace-ri.eu/HLST>.
- [26] https://www.prace-ri.eu/IMG/pdf/D5.2_5ip.pdf.
- [27] <https://www.etp4hpc.eu/cppp-monitoring.html>.
- [28] <https://aioti.eu>.
- [29] <https://www.eosc-portal.eu/>.
- [30] <http://www.bdva.eu/>.
- [31] <https://www.exascale.org/bdec/>.
- [32] <https://exdci.eu/collaboration/coe>.
- [33] <https://www.focus-coe.eu/>.
- [34] <https://eurohpc-ju.europa.eu/documents/>.
- [35] <https://ec.europa.eu/digital-single-market/en/news/19-proposals-selected-develop-world-class-supercomputing-ecosystem-europe>.
- [36] https://eurohpc-ju.europa.eu/closed_calls.html.
- [37] <https://eurohpc-ju.europa.eu/participate.html>.
- [38] <https://ec.europa.eu/digital-single-market/en/news/eurocc-and-castiel-two-new-projects-boost-european-hpc-knowledge-and-opportunities>.

- [39] <https://prace-ri.eu/cern-skao-geant-and-prace-to-collaborate-on-high-performance-computing/>.
- [40] <https://fenix-ri.eu/>.
- [41] <https://www.hpc-portal.eu>.
- [42] <https://www.skatelescope.org>.

Florian BERBERICH works for the PRACE Project Management Office at Jülich Supercomputing Centre (JSC) since 2008. He obtained his Ph.D. in physics at the Technical University of Dresden in 2002. He had worked as Post-Doc at the European Synchrotron Radiation Facility, France before he became the assistant to the Board of Directors at Forschungszentrum Jülich in 2004. Currently he is the project manager of the PRACE-6IP project, PRACE Council Secretary and a member of the Board of Directors of PRACE aisbl.

Janina LIEBMANN works for the PRACE Project Management Office as Project Assistant for the PRACE-6IP project at JSC since 2017. She finished her education as an industrial business management assistant at the EWE TEL GmbH, Germany, in June 2011. In 2016 she worked as a secretary at the Foundry Institute, RWTH Aachen.

Veronica TEODOR is working for the PRACE Project Management Office at JSC, since 2012. She finished her law studies at the University of Transylvania Brasov, Romania in June 2004 and multilingual communications at the University of Applied Science Cologne, Germany in January 2009. Currently, she is the task leader for the organisational support for PRACE 2 development in the PRACE-6IP project.

Jean-Philippe NOMINÉ joined CEA HPC division in 1992, where he held different managing positions in HPC software development. He has been involved in PRACE since its preparation in 2007 and has coordinated CEA efforts in all PRACE PP/IP projects. He was a member of PRACE aisbl Board of Directors in 2010–2011. He was then ETP4HPC Office Manager between 2012 and 2019 and now he is a member of ETP4HPC Steering Board, and of EuroHPC Research and Innovation Advisory Group (RIAG). At CEA he manages HPC strategic collaborations (EU and international). He graduated from Ecole Polytechnique (engineer degree) and obtained his Ph.D. from Université Pierre-et-Marie-Curie (Paris, 1991).

Oriol PINEDA is Director of Peer Review in PRACE, the Partnership for Advanced Computing in Europe, and Senior Project Manager at the Barcelona Supercomputing Center in Spain. He is responsible for monitoring the review process to access PRACE HPC resources and for the management of the impact assessment methodology of PRACE. He has an academic background, with a degree in organic chemistry, an M.Sc. in experimental chemistry and a Ph.D. in computational chemistry, from the University of Barcelona, Spain.

Philippe SEGERS is overseeing GENCI's contributions to the PRACE Implementation Projects, he is a member of the Board of Director of PRACE aisbl, Management Board of PRACE-nIP and PPI4HPC, and Technical Board of PRACE-nIP. He holds an Executive MBA from NEOMA Business School and has an academic background in numerical modeling of theoretical physics (University Paul Sabatier, Toulouse and University of Quebec at Montreal – UQAM). He began his career at the EC Joint Research Centre (Ispra, Italy) in 1999, moved to the scientific software industry and spent ten years in Program Management of R & D for EC and NATO projects. He was leading PRACE-3IP work-package on Pre-Commercial Procurement (PCP), he is a co-leader of PRACE-nIP work-package on the organisation of infrastructure and stakeholder management. He was representing PRACE within EOSCpilot project, and he is leading the GENCI contribution to the Connecting Europe Facilities (CEF) project AQMO on air quality.

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 validation¹.

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 communities²:

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 $O(47\text{ M})$, 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 appli-

cable. 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):

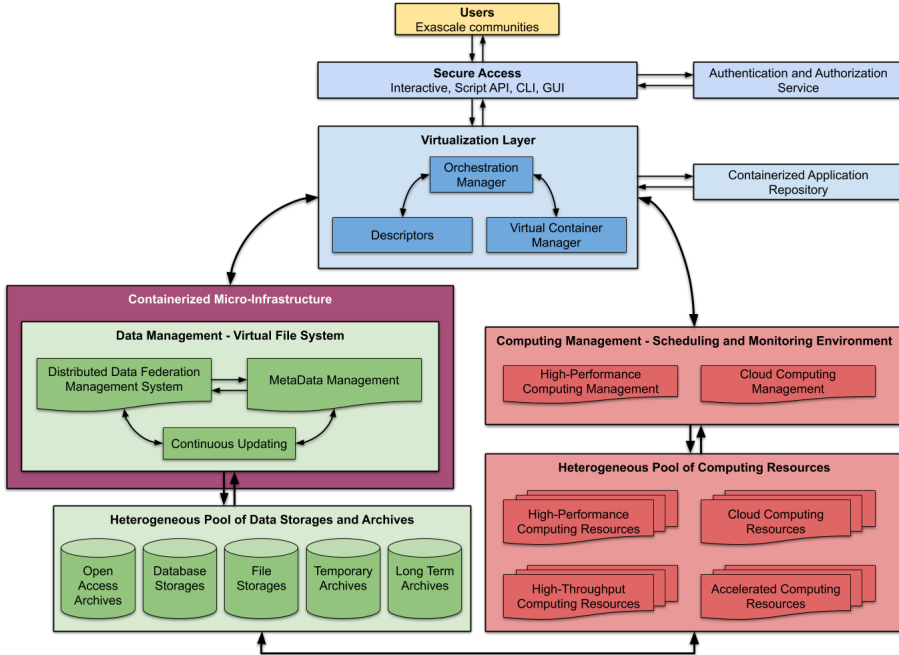


Figure 1. Reference exascale architecture

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-infrastructure³ 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-

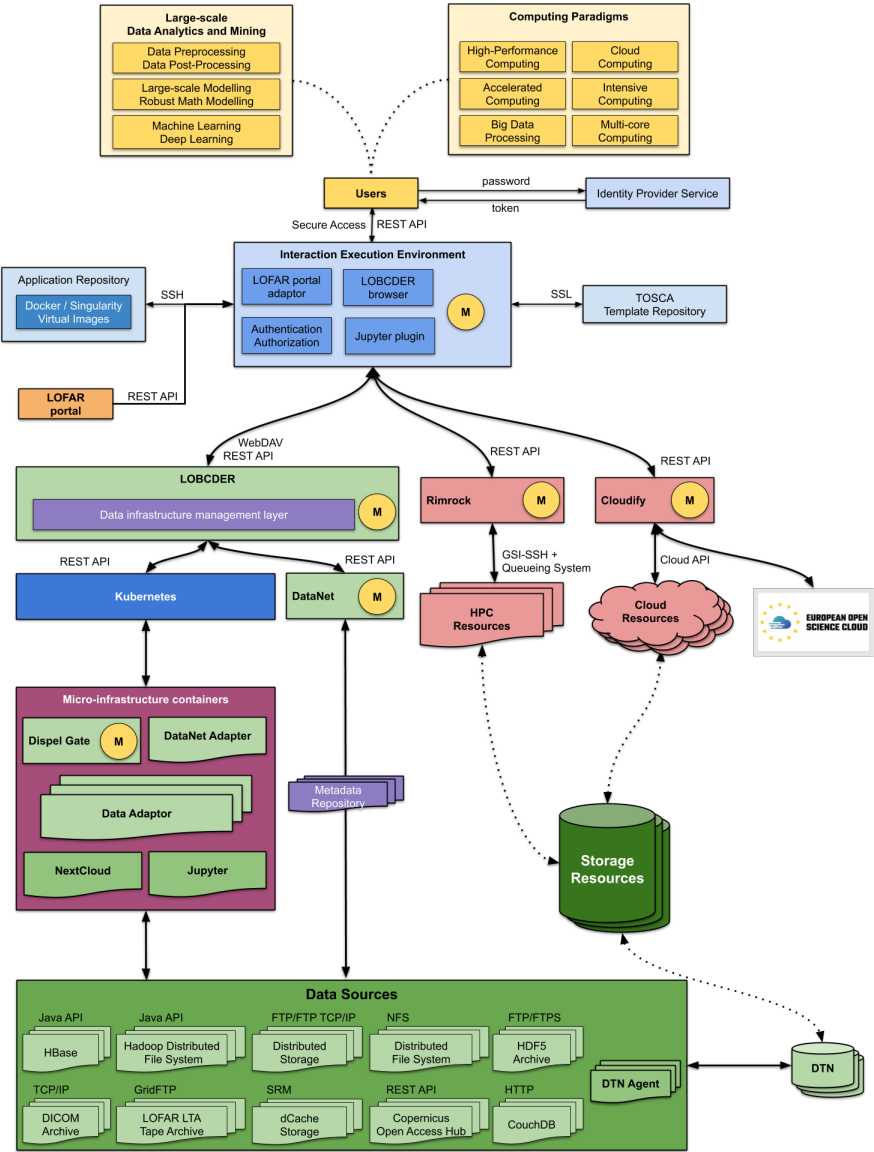


Figure 2. PROCESS architecture. (The yellow “M” token represents connection with a monitoring system. The PROCESS platform uses Zabbix as a monitoring agent.)

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 Cloudify⁴ for managing of cloud resources.

Cloudify can deploy VMs on any sites with OpenStack⁵ middleware, including sites in EOSC-Hub Federated Cloud infrastructure⁶. 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 project⁸. 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:

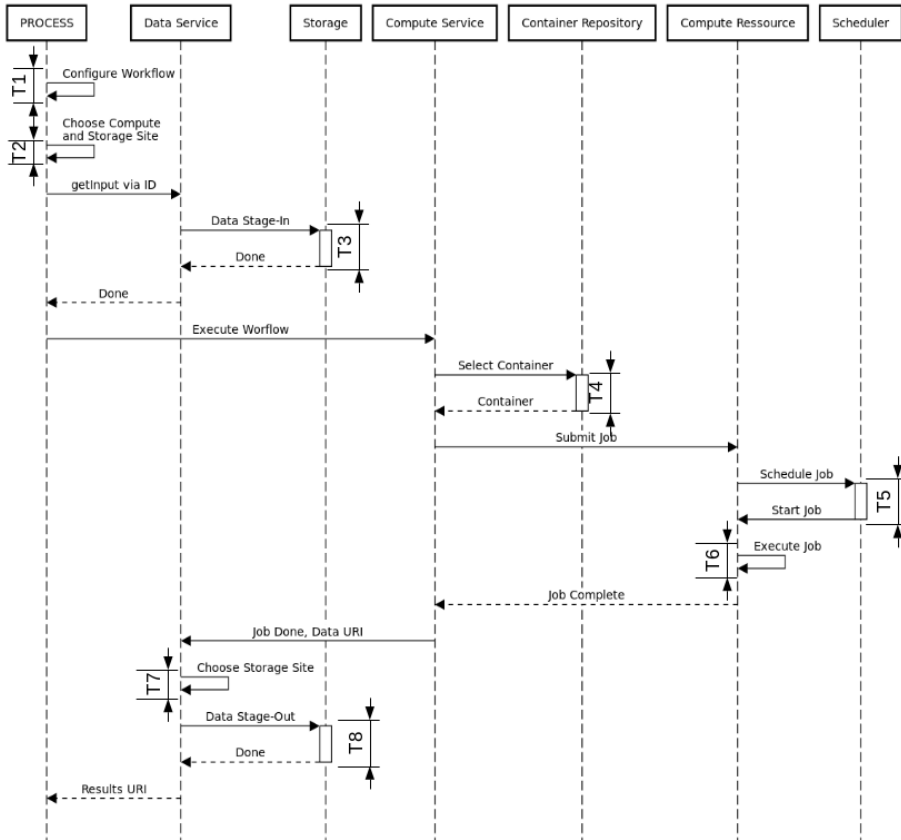


Figure 3. Sequence diagram describing the steps involved in execution of a typical application scenario

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 4a)). 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 4b)). 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 4c)). 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.

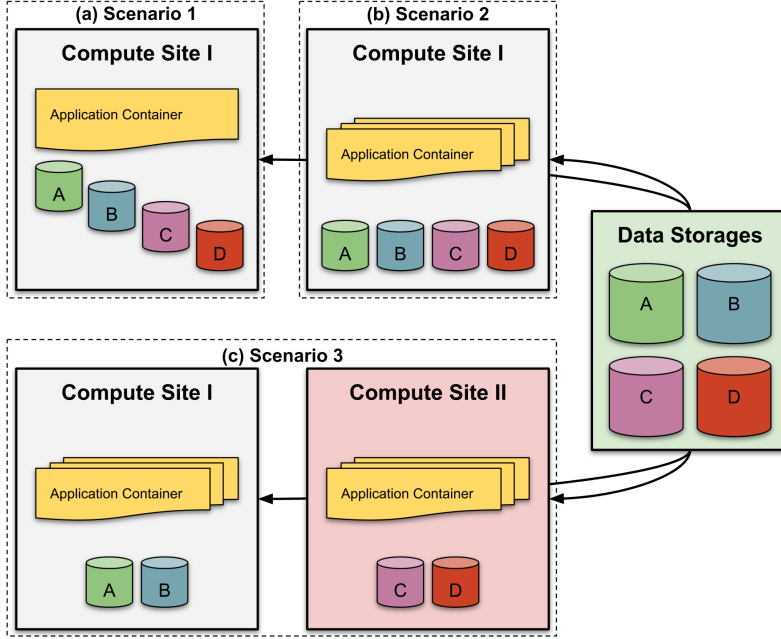


Figure 4. Three measurement scenarios: a) Single container – single site, b) Multiple container – single site, c) Multiple container – multiple site. In all three scenarios Stage-In and Stage-Out will downscale the system overall performances, unless we address the data transfer over a wide area network.

5.2 Runtime Composition

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

$$Runtime = Overhead + DataTransfer + Scheduling + ExecutionTime,$$

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

$$DataTransfer = T3 + T8,$$

$$Scheduling = T5,$$

$$ExecutionTime = T6.$$

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

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

The *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

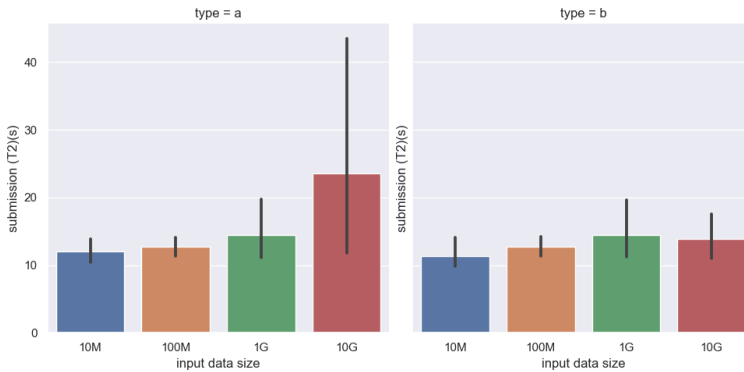


Figure 6. Submission delay in IEE batched by the input data size

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.

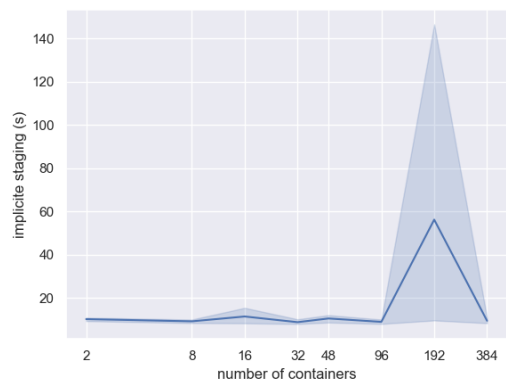


Figure 7. Implicit staging overhead behaviour in the PROCESS production prototype

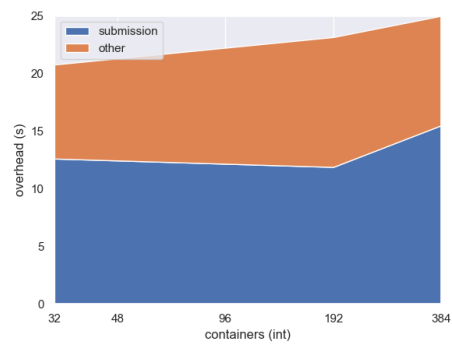


Figure 8. Overall overhead (T2+implicit staging) behaviour in the PROCESS production prototype

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

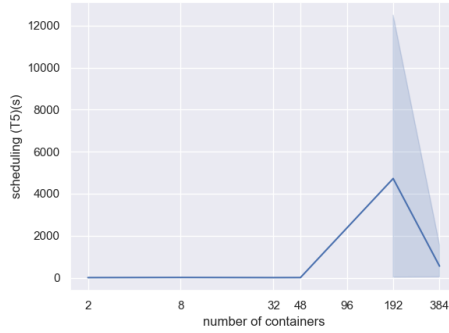


Figure 9. PROCESS scheduling overhead measurements from IEE

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

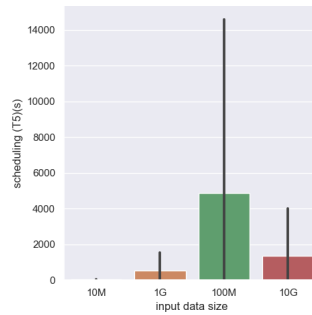


Figure 10. Scheduling delay measurements in the PROCESS production prototype batched by input size

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.

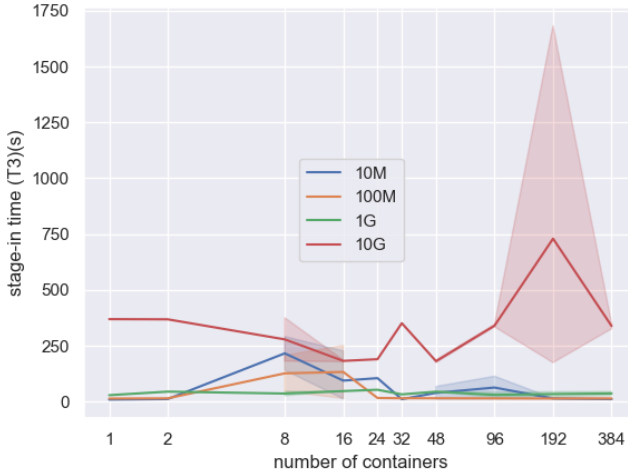


Figure 11. Staging-in measurements in the IEE production prototype

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: $overhead(o) = 0.011 * (number\ of\ containers(c)) + 20$) of the data shows a very slow variation of the overhead in function of the number of containers, which is very important for PROCESS scalability. Although the very small value of the slope implies that the increase is very slow, so we can consider this overhead as practically constant and independent of the number of containers. To put this in context, we can confidently assert that the overhead of processing the entire LOFAR LTA archive (around 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.

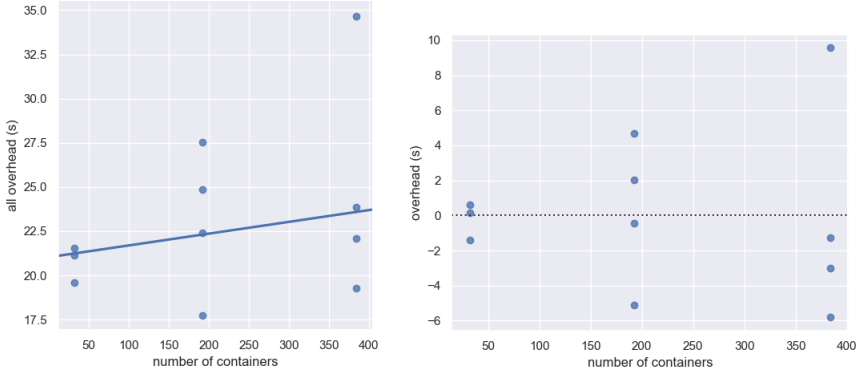


Figure 12. PROCESS T2 overhead models

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 $T3 = 0.032 * M + 39$ where M is the size of the input data in MB.

According to the linear model, it would take on average about 359s 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.

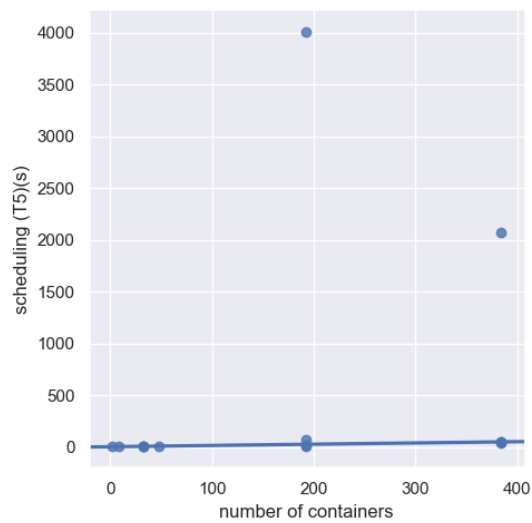


Figure 13. PROCESS scheduling overhead models in the IEE production prototype

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

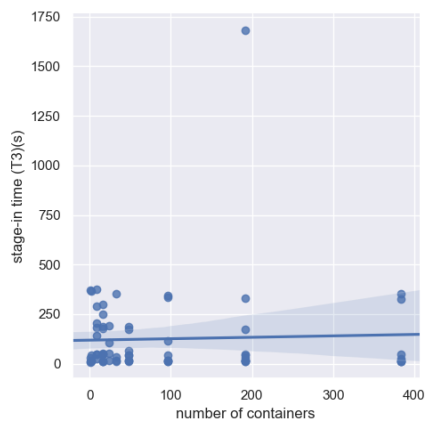


Figure 14. PROCESS staging-in delay model in IEE

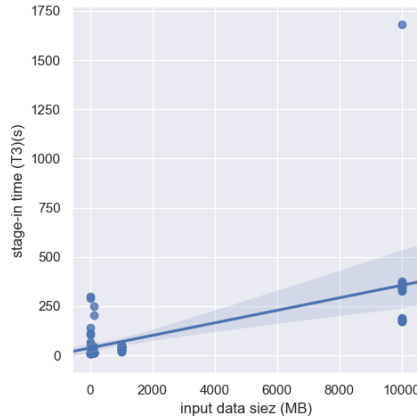


Figure 15. PROCESS staging-in delay model in IEE

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 inter-connection 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-STEEL, 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.—ŠELEN, 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.—BUYA, 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.—VANHOUCHE, 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 BOBÁK 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 INFOS-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 hydro-meteorology. He has over the years participated in numerous FP5, FP6, FP7, H2020 and national research projects and produced over 80 scientific publications.



Viet TRAN is Senior Researcher at the Institute of Informatics, Slovak Academy of Sciences (IISAS) 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.

COOKERY: A FRAMEWORK FOR CREATING DATA PROCESSING PIPELINE USING ONLINE SERVICES

Mikołaj BARANOWSKI

*Multiscale Networked Systems (MNS), Institute of Informatics
University of Amsterdam
Amsterdam, Netherlands*

Adam BELLOUM

*Multiscale Networked Systems (MNS), 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*

Reginald CUSHING, Onno VALKERING

*Multiscale Networked Systems (MNS), Institute of Informatics
University of Amsterdam
Amsterdam, Netherlands
e-mail: {r.s.cushing, o.a.b.valkering}@uva.nl*

Abstract. With the increasing amount of data the importance of data analysis in various scientific domains has grown. A large amount of the scientific data has shifted to cloud based storage. The cloud offers storage and computation power. The Cookery framework is a tool developed to build scientific applications using cloud services. In this paper we present the Cookery systems and how they can be used to authenticate and use standard online third party services to easily create data analytic pipelines. Cookery framework is not limited to work with standard

web services; it can also integrate and work with the emerging AWS Lambda which is part of a new computing paradigm, collectively, known as serverless computing. The combination of AWS Lambda and Cookery, which makes it possible for users in many scientific domains, who do not have any program experience, to create data processing pipelines using cloud services in a short time.

Keywords: Function-as-a-service (FaaS), serverless computing, AWS Lambda, domain specific languages (DSL)

1 INTRODUCTION

Cloud computing is getting a more and more prominent factor in life. People use cloud services every day, sometimes without even knowing. Cloud services come in a variety types like storage services, computation services, video streaming services and much more. Dropbox [1] and Google Drive [2] are well-known examples of cloud storage services that are widely used, with Dropbox claiming to have 600 million users in 2020 [3]. On the other side, more people tend to use cloud services for business purposes. Using cloud services, it is easier to collaborate across geographically distributed locations. In the Harvard Business Review: Cloud Computing Comes of Age [4], it is stated that cloud software greatly reduces the implementation time and it does not need a big up-front investment. It is also stated that a cloud provider could have an application up and running in five weeks, contrary to the 18 months that it would take according to the IT-business. On the other hand, the same review also shows that security of these cloud services is still the biggest barrier. For a number of potential cloud service users from various scientific disciplines like life sciences, health research, humanities, social sciences, environmental science, or earth science taking advantage of these cloud services has become more difficult and complex due to the many programming languages available and the documentations that accompany the APIs (Application Programming Interfaces) being rather technical. Besides not all scientists can afford to setup their own server; hence the cloud approach provides a very cost effective solution. They can buy some computing time or storage somewhere in the cloud, which will be cheaper at the beginning as opposed to buying a server. However, running a service on cloud resources comes with some additional problems. There is a need to build a software infrastructure to handle all the requests your application will receive. This is a long and tedious job that not everybody can do. In order to make programming using cloud services easier, Cookery has been developed in the context of Ph.D. research work at the University of Amsterdam [6].

Cookery framework allows users to develop applications that can connect features of multiple cloud service providers together. The Cookery approach is focused on programming distributed systems through a domain specific language. This approach is similar in terms of functionality to IFTTT (If This Then That) [5].

IFTTT is a web-based service that allows its users to create chains of conditional statements [25]. These chains contain triggers from cloud services that can invoke actions in other cloud services. Thus, IFTTT can be used to automate a web-application task. It will be useful to have the ability to extract, combine and exploit the best features of their favorite online services. With this ability users can develop applications that can automate their tasks, extend usability or simply give the user easier access to their data. In order to achieve this goal, a proof of concept will be developed in which the Cookery architecture is designed for optimal combination of functionalities of multiple cloud service providers. Important features of this architecture are the cloud service provider APIs, a server to handle authentication, Jupyter Notebook for user interaction and of course the Cookery kernel implementations itself.

Cookery enables scientists to combine multiple cloud applications in an easy way. On top of that, Cookery uses its own Domain Specific Language (DSL) to make it more accessible for users without any programming experience. The Cookery system comes with a couple handy features to make using cloud application seamless and easy to use, among interesting:

1. Cookery is integrated with the Jupyter notebook,
2. it implements the OAuth 2.0 protocol for a convenient authorization, and
3. it supports the use of AWS Lambda functions.

2 COOKERY

Cookery is a framework to make programming with other cloud applications easy [6]. Cookery makes it possible to combine cloud services using a high-level language. This high-level language, or Cookery language, has the same syntax as English, which makes it possible for users without any programming experience to easily create applications pipeline. A closer look at Cookery shows that it is actually composed of three layers, which can be seen in Figure 1.

- The first layer (Layer 1) is used by a user to create Cookery applications using the Cookery DSL language. In this layer data processing pipeline can be defined and modified, in this paper we will refer to the data processing tasks as “activities”. This is the highest abstraction level, which requires the least programming skills.
- The second layer (Layer 2) is for developers and, as opposed to the previous layer, this layer makes use of the Cookery Domain Specific Language (DSL). Layer 2 is used for defining and modifying actions, subjects and conditions. This layer abstracts the infrastructure details from application domain programmers.
- The third and last layer (Layer 3) is the Cookery backend and is also intended for developers. At this level, developers can implement protocols, which are for the activities and data, and communication with execution environments. This is the most lower layer offered by the Cookery framework.

Programming a Cookery layer is composed of simple statements called activities that will be executed by the framework. These activities consist of other elements, namely variables, actions, subjects and conditions. These elements all have their own role within the Cookery language. Variables are optional. They help to keep track of results of activities that could be referenced later using the labels; this feature helps to create data flows. An action refers to its implementation in the Cookery DSL and it represents remote operations. Subjects represent the input or output of an application, also known as remote data. They can, for example, be used for retrieving data from a cloud service. Both actions and subjects can be followed by arguments and both implementations are divided between all three levels. Conditions are used with keywords, like if or with, to separate them from the rest of an activity. Those are routines defined in Cookery DSL and are meant to transform data before it is passed to an action. This data can be retrieved in different ways, including from a remote location in a subject or from a variable. Layer 2 helps to develop the Cookery DSL, which allows Layer 1 users to use appropriate actions, subjects and conditions (Cookery elements). These elements all consist of a name, a regular expression and a Python routine. Cookery comes with a toolkit in order to make things easy. The toolkit enables a user to execute Cookery applications, generate new projects and evaluate expressions. More details about the Cookery language can be found in [6].

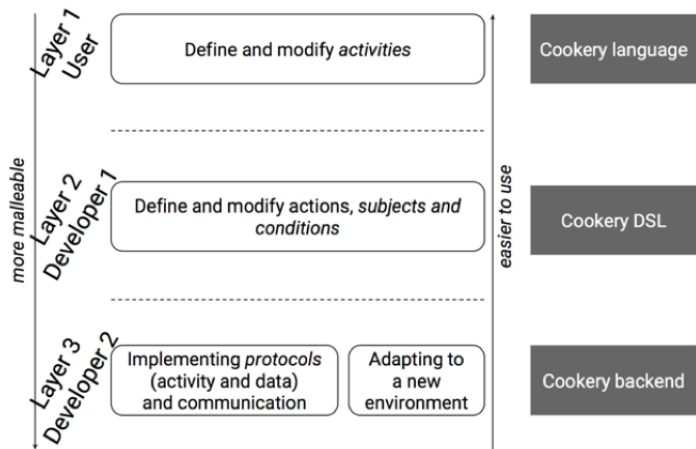


Figure 1. User roles and layers in Cookery [6]. The first layer is for developing applications, with Cookery language, the second layer is for defining actions, subjects and conditions using the Cookery DSL and the third layer is the back-end where protocols are implemented.

It can also provide more sophisticated features as code completion and transformation of rich objects (e.g., graphs). For languages that have Python bindings

(such as Cookery) the implementation of a kernel is much simplified using a kernel wrapper [26] which reuses the kernel machinery in IPython to create new kernels.

In the next subsection we present how Cookery systems implement the authentication and enable the use of AWS Lambda functions.

2.1 Authentication

Because Cookery Layer 1 users create data pipelines that can connect features of multiple cloud service providers together, these services often require third party access and involve payment. One of the main principles behind Cookery is that the user of Cookery application pays for its execution, not the application developer. While it makes a fair model for ensuring application reusability, it requires addressing a non-trivial issue of providing someone's resources to already existing application. Among the commonly used authentication methods OAuth 2.0 is one of the best option to enable third party access. The OAuth protocol provides a generic framework to let a resource owner authorize a third party that wants to access to its resources [14]. The widely used OAuth protocol is not yet fully without flows [15], however it does provide the user with the best combination of convenience and safety. OAuth does not expose the user to third party root access-right threats, and keeps the user in control over granted permissions. At the same time it brings a very convenient experience to the user because a minimal action from the end-users is required. Cookery takes a role of an OAuth client, it requests authorization from remote services and stores granted token for future use. A token is a cryptographically signed piece of text that is handed by the service to the client. Any future client to service communication can be done with the token. The service validates the token on requests which authorizes the service call. These limited time token can be given by the user to third party services so that they can access your resources on your behalf. The user and the service can decide to revoke the token thus blocking third party access. It can be successfully used in very different use-cases presented in Section 3. We decided not to make OAuth an essential nor required part of Cookery; it can be optionally used by layer 3 developers. Figure 2 shows how Cookery achieves OAuth authentication, Jupyter Notebook interacts with programming languages through the kernel. It acts as a middleman; it executes code given by Jupyter Notebook (provided by a user), executes it and handles results by transforming it to the form required by the target service.

2.2 Serverless Computing – AWS Lambda

Amazon Lambda is a service that provides serverless application deployment, many cloud service providers such as Google Cloud Functions [27] and Azure Functions [28] have FaaS in their portfolio. However, due to the programming language limitations of the other two (Google to JavaScript and Microsoft to JavaScript, C#, F#, and

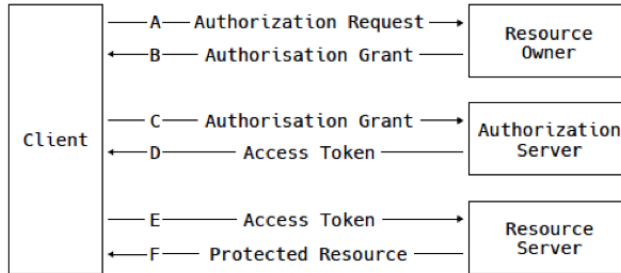


Figure 2. In this diagram of OAuth protocol flow [4], Cookery plays a role of a client regardless how it is deployed (Jupyter notebook, local script or on serverless computing platform such as Amazon Lambda). Resource owner can be understood as Cookery user while, from technical perspective, Resource owner can be the same entity as Authorization server. This, in turn, is a cloud service provider such as Google Cloud, Amazon AWS, etc. – usually sharing the same API as Resource Server.

scripting options: PHP, Bash, Batch, and PowerShell) we decided to experiment first with Amazon Lambda with Python 3.6. Our aim is to provide IFTTT [2] inspired functionality:

1. Deploy Cookery applications to Amazon Lambda with a single command. Deployment requires gathering all application sources including Cookery application, Cookery framework sources and all its dependencies within one zip file and upload it together with Lambda function specification (more details are given in Section 2).
2. Allow specifying periodic invocations. Periodic invocations require a trigger. Amazon Lambda supports several ways of triggering Lambda functions, we chose to use Amazon Cloud Watch Events [8]. It allows scheduling Lambda invocations with an arbitrary interval (more details are given in Section 2).
3. Enable connections to third-party cloud services. In order to enable connection to third-party cloud services, Cookery application has to be authenticated. If Cookery is used in an interactive mode – after invoking the code, user waits for results – all the missing authorization can be asked on demand. In case of serverless deployment, there are two components that have to be authenticated at different time:
 - Amazon Lambda service for application in order to deploy Cookery application in Amazon Lambda.
 - Protocols used in Cookery application during application execution on Amazon Lambda.

2.3 Scheduling

We will use AWS CloudWatch to schedule AWS Lambda functions, by creating rules. The scheduling function needs fewer parameters than the deployment function, which makes it easier to implement. A user will need to specify the period with which the function needs to be invoked. This period consists of a number and a period specifier like minutes, hours or days. Cookery adds some constraints to make it easier to work with a period. On top of that Cookery can also easily check if a rule already exists. The Cookery interface for scheduling Lambda functions first checks the given period and, with the rule name, checks whether this rule already exists. When this is the case, it just adds the Lambda function as a target to the rule. Because AWS regulates a limit of 5 targets per rule and a maximum of 100 rules, it is always possible to add every Lambda function to a rule. Cookery has to explicitly add permission for the rule to invoke the Lambda function; otherwise AWS will give an error. To add permission, the name of the rule is needed to acquire its ARN. A simple Cookery API Call allows the users to deploy and schedule a Lambda function using one API call, as shown in Listing 1.

```
cookery deploy
--function_name=...
--file_name=...
--handler=...
--every=... COOKERY_PROJECT_PATH
Options:
--function_name TEXT Name of the Function
--file_name TEXT Name of the file where the handler function can be found,
--handler TEXT name of the handler function
--every TEXT the interval to invoke the function m like 1,m, 1hm or 1d,
--help show this message and exit
```

Listing 1. Cookery command line API Call to deploy and schedule AWS Lambda functions

3 APPLICATION

To illustrate how Cookery authenticate to remote services, we will use a very simple proof of concept use case which consists in implementing a GitHub monitor that takes a repository property (in this case commits) and monitors it for changes (Figure 3). Whenever a change is detected, it triggers an action (email notification). The proof of concept application works with the Jupyter notebook, which is a web application itself. With this proof of concept application users can easily automate a GitHub related notification process.

The access to the repository is achieved by the implementation of an OAuth authentication, which is explained in detail in Figure 2. This implementation gives the user a convenient and secure authentication process. The monitoring works via periodic calls by a recursive function. This function compares the newest

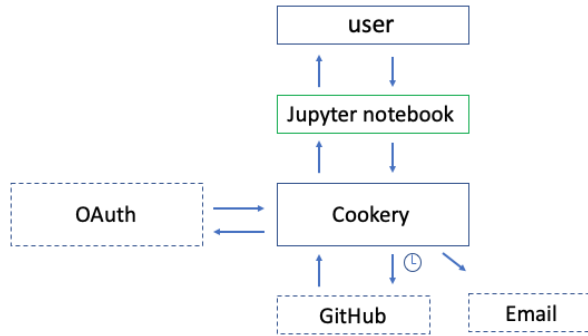


Figure 3. Scheme of the proof of the concept application GitHub monitor. The complete code implementing the GitHub monitor in GitHub [30]

commit with the latest monitored commit. If there is a difference between these two commits, an action is triggered. Because of the interval base of the monitoring mechanism, it could happen that two commits are made in one interval. When this happens only one notification gets send. The chance of this happening can be reduced by decreasing the interval. However, this will increase the server load.

Another variant of the GitHub monitor has been implemented using AWS Lambda [10]. This version of GitHub monitor application makes use of the GitHub REST API to get all the information while being authenticated with a GitHub personal access token. The scopes of this token can be configured by the user, like giving access to all private repositories, but not being able to delete them or to change user data. Making authenticated requests gives the opportunity to send 5000 requests per hour to the API, while making unauthenticated requests only gives us 60 requests per hour. This program first checks if the authenticated user has access to the given repository. This is done by sending a request using Python library *urllib*. The request is also needed to authenticate to GitHub by adding a header with the personal access token to it, which can be seen in Listing 2.

```
cookerydef
make_request (url):
    request = urllib.request.Request(url)
    request.add_header("Authorization", "token"+github_token)
    response = urllib.request.urlopen(request)
    data = response.read().decode("utf-8" )
    return data
```

Listing 2. The function to make requests to GitHub with authentication using urllib

Implementation of the GitHub monitor with using AWS Lambda functions. The complete code implementing the GitHub monitor using AWS Lambda is available

```

cookery deploy
    -- function_name = "github_monitor"
    -- file_name = "github_monitor"
    -- handler = "github_monitor.handle"
    -- every = 5mn "https://github.com/mikolajb/cookerylambda/blob/master/
    github_monitor/github_monitor.py"

```

Listing 3. Cookery call to deploy and schedule the GitHub monitor AWS Lambda Function

in GitHub [31]. When a GET request is sent to <https://api.github.com/user/repos>, a response in JSON format is sent back, which can easily be deserialized to a Python object, containing the list of accessible repositories. This way we can examine every repository and check if the target repository is among them. A second more specific request is sent to get the list of all the commits, sorted by time, with the last commit first. In the response, we can see how many additions and deletions are made, and what the changes are in every file. This information will then be put into the body of an email, which is sent with Gmail using the Python SMTP library. The authentication of Gmail is done with an application key, just like with GitHub.

As mentioned before, the Lambda functions are stateless and terminate after 5 minutes. This means that we cannot use any variables after termination, except when we save them to a database or communicate them back. The way the GitHub monitor works is thus the simplest. We have the same problem with giving the repository to check. This information will be lost after termination and needs to be given every time the function gets invoked. This is where the environmental variables come in. They can be given to a Lambda function as key-value pair and they will be the same for every invocation when deployed. This makes it also possible to reuse the GitHub monitor to check another repository, without changing the code.

3.1 Finding Life Trends Using Data Analysis in Cookery

The use case shows how to create a data pipeline in Cookery [11]. The use case is about analyzing Gmail in order to visualize life trends. As pointed out in Section 2, Cookery framework offers a separation between the creating of general function by developers (Layer 2) and the creation of user-specific programs by the end-user (Layer 1). The layers reduce the amount of code that the end-user has to write, but still allow flexibility. Layer 1 is where the end-user will be interacting. The user will combine already defined activities to create a Cookery program. The use case aims to create in Cookery a data processing pipeline similar to Stephen Wolfram “The Personal Analytics of My Life” [12]. In our case the personal analytics is limited to analyzing the emails of one of the authors. The data analytics pipeline combines a number of online services providing the initial data Gmail service, and Google

analytics service. A visual representation of the data flow for the project is shown in Figure 4. This means that Cookery and the data pipeline within the need to handle data from multiple sources, some of which need to be stored to a file or be kept in the program memory, and give visual results back to the user.

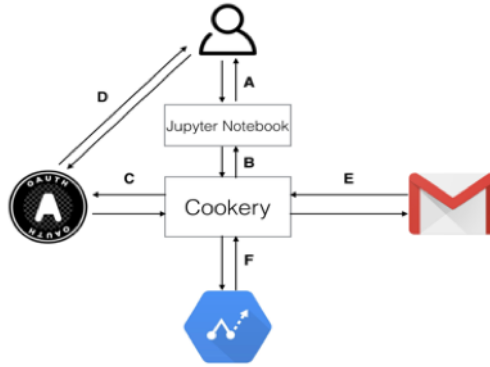


Figure 4. Finding life trends using data analysis in the Cookery data processing pipeline which is composed of two third-party services (Gmail, Google analytics)

The data pipeline can be broken into three steps:

1. The authentication step using OAuth protocol (Figure 4 – C): The first step is to register the application at Google in order to obtain the credentials. After the registration, it is possible to download the “client secret”, which is stored on client side; this part is similar to the previous case using the GitHub monitor. Now that the application is known at the OAuth authentication server, we can start writing code to interact with the servers. In this case the application is limited to read emails and access the prediction API. The credentials are stored, so this verification needs to be done only once. Unless the scopes get changed, the credentials expire or the user retracts the permission.
2. The retrieval of individual emails from the server (Figure 4 – E): In this phase, it is possible to specify the characteristics of the request to read the emails, an example could be the label “INBOX” or “SENT”. In the list of arguments of the request are the IDs of individual messages that were sent, which can be retrieved by their ID. For the purpose of this example, we are interested in the body of the email, the time and the date and the list of recipients, which are formatted as “Alias – Email address” and grouped by their address field (TO, CC, BCC).
3. Implement the machine learning API (Figure 4 – F): This was done in a similar way as with the Gmail API. We focused on the pre-trained models of the API. These models are used to analyze and predict the language of the emails. Google made the models based on their data set. A body of a message is sent to the servers, analyzed and returned to the Cookery system. The possible outcomes

of the semantics analysis are positive, negative or neutral. The text of each email is categorized as either English, Spanish or French. The complete code implementing the data analysis workflow for the email analysis case is available in GitHub. The user can specify how many months of email they would like to analyze. All the incoming and outgoing emails are analyzed. For the incoming emails we are only looking at the date and time and the number of emails. For the outgoing emails we are interested in the date and time, the participants, number of emails, semantics and languages. All this information was stored as an object per email in a dictionary.

After Cookery finished analyzing emails, we visualized the dictionary of analyzed emails. The visualization was done outside Cookery using Matplotlib [13]. The plot is based on the last six month of author's personal emails which contain 470 outgoing and 1 200 incoming messages. The results based on the time stamp of the email and the number of emails are shown in Figure 5 a), and the results of the sentiment analysis are in Figure 5 b).

4 RELATED WORK

This research is focused on the combination of multiple systems; it has a lot of similarity with IFTTT (If This Then That) [14]. IFTTT is a web-based service that allows its users to create chains of conditional statements. These chains contain triggers from cloud services that can invoke actions in other cloud services. Thus, IFTTT can be used to automate web-application tasks. Examples are uploading photos to Dropbox that were received by email or automatically uploading the same content to multiple social media streams. IFTTT is focused on simplicity and therefore only supports one trigger action pair. Other IFTTT alternatives Zapier [16], and Microsoft Flow [17] differ from that approach and offer support for longer chains of trigger/actions pairs. Apart from offering longer chains, Zapier and Microsoft Flow use the same concept for workflow automation as IFTTT. Cookery differs from these services by its ability to implement any data transformation, because it can be extended with all the functionalities that Python has to offer.

In the article "Serverless Computation with OpenLambda" [18] the authors show that we have reached a new stage in the sharing model with FaaS. Technology has progressed from only sharing the hardware, which is done with virtual machines like VMWare, sharing the hardware and the operating system, as seen with containers like Docker, to sharing the runtime of a system. The handler is started in a container, which can only be used by the handler itself. Although multiple containers run in the same run-time, communication between containers is not possible. Other functions would then be able to intercept your functions and gain access to valuable information. On the other hand, a user will have to recognize some places where performance issues can arise. The readiness latency, the time it takes to start, restart or resume a container, can have consequences for the overall performance [18]. And there are more like the number of containers per memory (container density), pack-

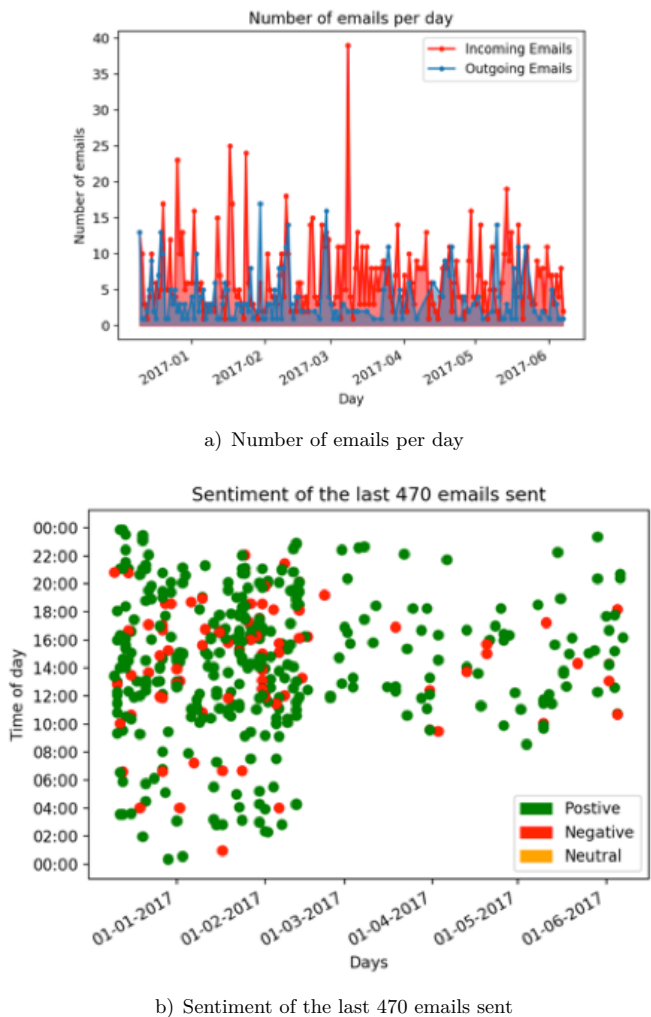


Figure 5. Outcome of the sentiment analysis of the inbox of the email account of one of the authors

age support, cookies and sessions. A study has compared the cost, performance and response time of different implementation architectures such as monolithic architecture, micro-service architecture operated by the cloud customer and micro-service operated by AWS Lambda. With the micro-service architecture a developer will try to develop an application as a suite of small services [19], which all run their own process. The results of this study show that a micro-service operated by AWS Lambda is up to 77.08 % cheaper per million requests than the other two methods, while

the response times are faster than the cloud customer operated micro-service architecture and about the same as the monolithic architecture [20]. There are multiple online platforms that offer FaaS. Some examples are Google Cloud Function [21], Microsoft Azure, AWS Lambda and IBM OpenWhisk [22]. AWS Lambda is chosen for this project, because it is the only one to offer the service in combination with Python.

5 CONCLUSIONS AND FUTURE WORK

The goal of the Cookery framework is to enable scientists with little programming background to define a data analysis pipeline and execute it on online services provided by multiple cloud providers. To achieve this goal Cookery had to simplify process of the authentication and authorization, allowing the users to focus only on creating the data processing pipeline. In this paper, we described extension to the Cookery systems that allow for connections with cloud services using the OAuth 2.0 protocol. We used the “life trends” example to demonstrate how to create a data processing pipeline in Cookery using online services namely the Gmail service, and Google analytics. When developing the “life trends” example using cloud services, the only limitations we faced were related to the fact that we used a free version of the Google analytics, the machine learning API has a “User rate limit”, which might have an impact on the performance of the time critical data processing pipeline. In the “life trends” example, we only made use of services from a single service provider namely Google.

Our next goal is to build data processing pipeline using service from different providers like AWS Amazon and Microsoft Azure. The first step toward this goal is described in the paper; we have developed an extension to use AWS Lambda services. In the future Cookery will be extended with more services from different cloud providers, to create a broader framework and to enable more developers to create applications. For example, an interesting extension would be with AWS DynamoDB [23] or AWS RDS (Relational Database Service) [24] to make it easier to create and manage databases using Cookery. When we combine databases with the functions of AWS Lambda, we can create more complicated applications and deploy them using Cookery. This also means that the toolkit of Cookery can be extended with more services and functionalities in future projects.

An important extension, we are currently investigating as a potential easy to develop light web application is the recently established Function-as-a-Service (FaaS). In the FaaS approach a user only runs a function on an external server. This is also called serverless computing, because you do not need a server for your application anymore. You basically have the function running on a server in the cloud. A user of the application provides the input and the function returns the output. This makes it possible for smaller businesses to develop their own application without buying servers. Developers also do not need a system administrator to maintain the servers, they do not need to write a complete infrastructure that can scale with the

demands of the applications and they do not need to handle all the administration. So basically, applications can scale up rapidly without needing to start new servers.

Inline with the FaaS concept we are looking at integrating this work with the idea of micro-infrastructures [29]. A micro-infrastructure is a dedicated network of application specific containers that are hosted on Kubernetes clusters. The containers expose functionality as FaaS so it is a natural coupling with Cookery. The added benefit of a micro-infrastructure is to define your own complex routines, package them in containers and expose them as FaaS. These can be placed closer to the data, in cases where the data is not on the public cloud but on HPC resources.

This can be developed further into a full-stack approach. An ecosystem of development tools would be used to capture isolated functionality at each level of the technical stack. The orchestration of the entire stack, composed of these isolated functionalities, can then be expressed using the DSL. Including, but not limited to, setting up (private) networks, data transfers, running compute routines in the cloud or on HPC clusters, and invoking remote web services. In the spirit of Cookery, each tool would provide a different abstraction level, and targets a specific level of the technical stack and/or user role. This will not only make the life easier of the scientists, operating at the highest level of the stack, but also supports efforts by engineers at the underlying levels of the stack. The low to non-existing coupling between isolated functionalities also facilitates reusability and maintainability. Moreover, functionality can easily be swapped out for a different version or an alternative implementation. To ensure the valid coherence of a application, composed of isolated functionality, the DSL can be extended with a type system. The type system will validate the application composition before it is executed, identifying interoperability issues at compile time. This prevents unnecessary costs, i.e. development time and resource usage expenses, caused by running invalid applications.

Acknowledgment

The authors thank Michael van Mill, Timo Dobber, and Dennis Kruidenberg – students at the University of Amsterdam who helped in implementing the three extensions of the Cookery framework presented in this paper.

REFERENCES

- [1] Dropbox. <https://www.dropbox.com/>.
- [2] Google Drive. <https://www.google.com/drive/>.
- [3] Dropbox Statistics, Users, Growth and Facts for 2020. <https://saasscout.com/dropbox-statistics/>.
- [4] Harvard Business Review Analytic Services. Cloud Computing Comes of Age. 2015.
- [5] UR, B.—HO, M.P.Y.—BRAWNER, S.—LEE, J.—MENNICKEN, S.—PICARD, N.—SCHULZE, D.—LITTMAN, M.L.: Trigger-Action Programming in the Wild: An Analysis of 200 000 IFTTT Recipes. Proceedings of the 2016 CHI Conference on

- Human Factors in Computing Systems (CHI'16), ACM, 2016, pp. 3227–3231, doi: 10.1145/2858036.2858556.
- [6] BARANOWSKI, M.—BELLOUM, A.—BUBAK, M.: *Cookery: A Framework for Developing Cloud Applications*. Proceedings of IEEE International Conference on High Performance Computing and Simulation (HPCS), 2015, pp. 635–638, doi: 10.1109/HPCSim.2015.7237105.
 - [7] VAN MILL, M.: *A Cookery Extension to Simplify Cloud Service Integrations*. Bachelor Thesis. University of Amsterdam, 2017, <https://esc.fnwi.uva.nl/thesis/centraal/files/f704994072.pdf>.
 - [8] Low-Level Clients. <http://boto3.readthedocs.io/en/latest/guide/clients.html>, visited on 04/20/2017.
 - [9] <https://aws.amazon.com/s3/>.
 - [10] DOBBER, T.: *Cookery in AWS Lambda*. Bachelor Thesis. University of Amsterdam, 2017, <https://esc.fnwi.uva.nl/thesis/centraal/files/f274795790.pdf>.
 - [11] KRUIDENBERG, D.: *Finding Life Trends Using Data Analysis in Cookery*. Bachelor Thesis. University of Amsterdam, 2017, <https://esc.fnwi.uva.nl/thesis/centraal/files/f628826658.pdf>.
 - [12] WOLFRAM, S.: *The Personal Analytics of My Life*. 2012, <https://writings.stephenwolfram.com/2012/03/the-personal-analytics-of-my-life/>.
 - [13] Python Package Matplotlib. <https://matplotlib.org>.
 - [14] YANG, F.—MANOHARAN, S.: *A Security Analysis of the OAuth Protocol*. 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), IEEE, 2013, pp. 271–276, doi: 10.1109/PACRIM.2013.6625487.
 - [15] LODDERSTEDT, T.—MCGLOIN, M.—HUNT, P.: *OAuth 2.0 Threat Model and Security Considerations*. Internet Engineering Task Force (IETF), 2013.
 - [16] <https://zapier.com/developer/documentation/v2/#what-is-zapier>.
 - [17] *Getting Started with Microsoft Flow*. <https://www.windowscentral.com/getting-started-microsoft-flow>.
 - [18] HENDRICKSON, S.—STURDEVANT, S.—HARTER, T.—VENKATARAMANI, V.—ARPACI-DUSSEAU, A. C.—ARPACI-DUSSEAU, R. H.: *Serverless Computation with OpenLambda*. Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'16), 2016, pp. 33–39.
 - [19] FOWLER, M.—LEWIS, J.: *Microservices: A Definition of This New Architectural Term*. 2014, <https://martinfowler.com/articles/microservices.html>.
 - [20] VILLAMIZAR, M.—GARCÉS, O.—OCHOA, L.—CASTRO, H.—SALAMANCA, L.—VERANO, M.—CASALLAS, R.—GIL, S.—VALENCIA, C.—ZAMBRANO, A.—LANG, M.: *Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures*. 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2016, pp. 179–182, doi: 10.1109/CCGrid.2016.37.
 - [21] Google Cloud Functions. <https://cloud.google.com/functions/>.
 - [22] IBM OpenWhisk. <https://developer.ibm.com/openwhisk/>.
 - [23] AWS DynamoDB. <https://aws.amazon.com/dynamodb/>.

- [24] AWS RDS. <https://aws.amazon.com/rds/>.
- [25] IFTTT: About IFTTT. <https://ifttt.com/about>.
- [26] Making Simple Python Wrapper Kernels. <http://ipython.readthedocs.io/en/stable/development/wrapperkernels.html>.
- [27] Google Cloud Functions Documentation. 2017, accessed: 11-Dec-2017, available at: <https://Cloud.google.com/functions/docs/>.
- [28] Microsoft Azure: Azure Functions. 2017, accessed: 11-Dec-2017, available at: <https://azure.microsoft.com/en-us/services/functions/>.
- [29] CUSHING, R.—VALKERING, O.—BELLOUM, A.—DE LAAT, C.: Towards a New Paradigm for Programming Scientific Workflows. 2019 15th International Conference on eScience (eScience), San Diego, USA, 2019, pp. 604–608, doi: 10.1109/eScience.2019.00083.
- [30] <https://github.com/mikolajb/cookery>.
- [31] <https://github.com/mikolajb/cookery/tree/master/cookery>.

Mikołaj Baranowski is Ph.D. student at the University of Amsterdam in the Multiscale Networked Systems (MNS) group. His research fields are new paradigms in computer language abstractions.



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



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



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

PERFORMANCE EVALUATION OF PARALLEL HAEMODYNAMIC COMPUTATIONS ON HETEROGENEOUS CLOUDS

Oleg BYSTROV, Arnas KAČENIAUSKAS, Ruslan PACEVIČ
Vadimas STARIKOVIČIUS, Algirdas MAKNICKAS
Eugenius STUPAK, Aleksandr IGUMENOV

Vilnius Gediminas Technical University

Saulėtekio 11, Vilnius 10223, Lithuania

e-mail: {oleg.bystrov, arnas.kaceniauskas, ruslan.pacevic,
vadimas.starikovicius, algirdas.maknickas, eugenius.stupak,
aleksandr.igumenov}@vgtu.lt

Abstract. The article presents performance evaluation of parallel haemodynamic flow computations on heterogeneous resources of the OpenStack cloud infrastructure. The main focus is on the parallel performance analysis, energy consumption and virtualization overhead of the developed software service based on ANSYS Fluent platform which runs on Docker containers of the private university cloud. The haemodynamic aortic valve flow described by incompressible Navier-Stokes equations is considered as a target application of the hosted cloud infrastructure. The parallel performance of the developed software service is assessed measuring the parallel speedup of computations carried out on virtualized heterogeneous resources. The performance measured on Docker containers is compared with that obtained by using the native hardware. The alternative solution algorithms are explored in terms of the parallel performance and power consumption. The investigation of a trade-off between the computing speed and the consumed energy is performed by using Pareto front analysis and a linear scalarization method.

Keywords: Cloud computing, parallel computing, haemodynamic flows, parallel performance analysis, energy consumption, bi-objective optimization problem

Mathematics Subject Classification 2010: 68M14, 68M20, 65Y05

1 INTRODUCTION

In spite of noticeable recent achievements in medicine and technology, cardiovascular diseases are one of the leading causes of death in the world [1]. The heart is a complex system governed by haemodynamics [2], structural dynamics and electromagnetics [3, 4]. The efforts to create fully coupled holistic models of the heart valves have not been applied to a clinical patient-specific base yet. Present-day in-vivo measurement techniques can only resolve large-scale features of the haemodynamic cardiovascular flows [2]. Despite the progress in the numerical methods and constantly increasing power of modern computers, the considered problem is still highly challenging, owing to complex moving geometries, intrinsic flow unsteadiness, very intense velocity gradients, simulation divergence and mesh dependent numerical solutions. Most of the performed haemodynamic analyses have been restricted to non-physiological flow regimes, simplified solution domains, laminar flow simulations and relatively coarse space discretization due to computational challenges [3]. Computational fluid dynamics (CFD) simulations [2, 5] of blood flow in geometries extracted from medical images seem to be well suited for the patient-specific analysis and are compatible with clinical routine [6]. The required level of detail makes the patient-specific haemodynamic simulations of heart chambers computationally very expensive [3]. Naturally, to establish quantitative links between the aortic valve flow patterns and cardiac disease, parallel computations have become an obvious option for significantly increasing computational capabilities. Software for complex biomechanical and haemodynamic computations is usually deployed as an HPC solution [7].

Cloud computing is a distributed computing paradigm that has recently gained great popularity as a platform for on-demand, high-availability and high-scalability access to resources. Generally, clouds provide three levels of services [8]: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS). Deployment of the software services (SaaS) for data preparation, high performance computation and visualization on the cloud infrastructure increases the mobility of users and achieves better exploitation because clouds feature flexible management of resources. Thus, flexible cloud infrastructures and software services are perceived as a promising avenue for future advances in the multidisciplinary area of haemodynamic computations, such as numerical analysis of the patient-specific aortic valve flows. However, cloud computing still lacks case studies and quantitative comparison of performance in the case of specific applications. Most of evaluations of virtualization overhead and performance of cloud services have been based on standard benchmarks [8], therefore, the impact of the numerical issues and algorithmic aspects of haemodynamic applications on the performance of parallel computations remains unclear. The growing demand for cloud services and modern computational needs results in the development of large-scale IT infrastructures, which cause a considerable increase in power consumption [9]. Power efficiency is a crucial factor in the cloud computing environment. Green cloud computing is cloud computing with the efficient use of power, which helps to reduce power consumption and carbon

emissions. Energy consumption varies significantly depending on the application, workload, scheduling strategy and virtualization overhead. Power efficiency can be reduced, when virtualization is used, contrary to physical resource deployment, to grant all application requests [10]. In cloud computing, the problem of energy-efficiency is still a challenge mainly because of the variety of applications that need to be processed on cloud infrastructures and the demand for high performance.

The present article describes the efficiency analysis of parallel numerical algorithms for performance- and energy-aware computations of the haemodynamic aortic valve flows carried out on Docker containers of the heterogeneous cloud infrastructure. The aim of the presented research is to demonstrate that efficient computations on heterogeneous clouds require the elaborate selection of numerical solution algorithm and domain decomposition method that are highly dependent on the considered application. Information provided by the synthetic benchmarks usually performed on clouds does not include all important factors and it is not sufficient for finding the best hardware setup. Therefore, the application specific tests need to be performed before the production runs to optimize the parallel and energy efficiency of computationally demanding applications. Other parts of the article are arranged as follows: in Section 2, the related works are overviewed and discussed, Section 3 describes the patient-specific aortic valve problem and Section 4 presents the hosted cloud infrastructure and the developed software services. The parallel performance analysis, energy consumption and solution of a bi-objective optimization problem are discussed in Section 5, while the concluding remarks are presented in Section 6.

2 THE RELATED WORKS

Cloud computing is becoming a natural solution to the problem of expanding computational needs due to its on-demand nature, low-cost and offloaded management [8]. For haemodynamic analysis, cloud computing and Linux containers can offer a convenient, scalable alternative to traditional methods of managing computational resources.

There are different implementations of cloud software that organizations can utilize for deploying their own private cloud. OpenStack is an open source cloud management platform delivering an integrated foundation to create, deploy and scale a secure and reliable public or private cloud [11]. Another popular cloud computing framework, Eucalyptus [12], implements infrastructure services enabling users to run and control virtual machine (VM) instances across a variety of physical resources. Cloud computing makes the extensive use of virtual machines (VM) because they allow workloads to be isolated and the resource usage to be controlled. Xen is primarily a bare-metal, type-1 hypervisor which can be directly installed in the computer hardware without the need for a host operating system [13]. Kernel Virtual Machine (KVM) [13] is a feature of Linux that allows Linux to act as a type 1 hypervisor, running an unmodified guest operating system inside a Linux process. Containers

present an emerging technology for improving the productivity and code portability in the cloud infrastructures. Container-based virtualization was initially viewed as a lightweight alternative to virtual machines. Rather than running a full operating system on virtual hardware, container-based virtualization modifies an existing operating system to provide extra isolation. Container runtimes, such as LXC [14] and Docker [15], largely abstract away the differences between the many operating systems that users run. In many cases, Docker container images require less disk space and I/O than the equivalent VM disk images due to the layered file system. Thus, Docker has emerged as a standard runtime, image format, and build system for Linux containers. HPC vendors have also begun integrating native support for Docker. For example, IBM has added Docker container integration to Platform LSF to run the containers on an HPC cluster [16]. Container computing has revolutionized the way groups are developing, sharing, and running software and services. Recently, container computing has gained traction in the HPC community through enabling technologies like Docker, Charliecloud [17] and Singularity [18]. Container runtimes, such as Singularity and Charliecloud, allow end-users to run containers in environments, where standard Docker tools would not be feasible. Charliecloud [17] employs the Linux user and mount namespaces to run industry-standard Docker containers with no privileged operations or daemons on center resources. Singularity [18] is a novel containerization technology that proposes quickly deployable and transferable containers without encapsulating an entire OS inside the containers images. Sauvanaud et al. [19] have investigated performance of big data applications based on Hadoop, performing scenarios on Singularity and Docker instances. UberCloud application software containers have provided ANSYS Fluids and Structures software [20]. However, it is hardly possible to provide precise guidelines regarding the optimal cloud platform and virtualization technology for each type of research and application [8]. Moreover, the performance is a critical factor in deciding whether or not containers are viable for scientific software.

The performance of virtual machines and lightweight containers has already received some attention in the academic literature because they are crucial components of the overall cloud performance. Seo et al. [21] have compared the performance of containers and virtual machines for non-scientific software stacks deployed in the cloud. In the research performed by Kačeniauskas et al. [22], the performance of the private cloud infrastructure and virtual machines of KVM has been assessed testing CPU, memory, hard disk drive, network and the software services for medical engineering. The measured performance of the virtual resources has been close to the performance of the native hardware measuring only the memory bandwidth and disk I/O. Di Tommaso et al. [23] have compared the performance of some commonly used genetics analysis software running natively and inside a container. Production load for scientific experiments carried out on Docker containers has been investigated by Mazzoni et al. [24]. Estrada et al. [25] have executed genomic workloads on the KVM hypervisor, the Xen para-virtualised hypervisor and LXC containers. Xen and Linux containers exhibited near-zero overhead. In the previous work of the authors [26], the performance of the developed software services for haemodynamic

computations was measured on Xen hardware virtual machines, KVM virtual machines, Docker containers and compared with the performance achieved by using the native hardware. Most of the discussed performance studies [23, 24, 25, 26] have found negligible performance differences between a container and a native hardware.

Han et al. [27] have performed MPI-based NAS benchmarks on Xen and showed that the measured overhead became higher when more cores were added. Strong and weak scalability of the alternative parallel solvers for the aortic valve flows has been examined in Rocks cluster [28], but virtualisation overhead and alternative domain decomposition methods have not been considered. A study [29] on the use of container-based virtualisation in HPC has revealed that Xen VM was slower than LXC container by roughly of the factor of 2, while a native server and LXC container had near-identical performance. Hale et al. [30] have shown that the performance of Docker containers, when using the system MPI library for parallel solution of the Poisson's equation carried out by FEniCS software, was comparable to the native performance. Mohammadi and Bazhiron [31] have performed the High Performance Linpack benchmark on cloud computing infrastructures managed by Amazon Web Services, Microsoft Azure, Rackspace, IBM SoftLayer and demonstrated that the performance per single computing core on public cloud could be comparable to modern traditional supercomputing systems. The authors of the present article have performed the initial MPI-based benchmarks [32] on virtualized resources of homogeneous OpenStack cloud infrastructure.

The pervasive use of cloud computing and the resulting rise in the number of hosting centres have brought forth many concerns including the cost of power, as well as peak power dissipation and cooling. One of the great challenges of cloud infrastructures is to manage system resources in an energy-efficient way. In computer infrastructures, energy-efficiency can be enhanced at three different levels [33], such as energy-efficient applications, energy-efficient hardware and power-aware resource management. Energy-efficient applications are developed using the energy-efficient algorithms [34], special lower level programming techniques, including dynamic voltage and frequency scaling [35], data reuse methodology [36], etc. However, manual tuning of application for higher energy-efficiency remains a time consuming and challenging task. Low-power CPU, memory and other components of energy-efficient hardware [37] can effectively support the static power management. The dynamic power management employs load balancing techniques [38] and power-scalable hardware components [35] to optimize energy consumption. Load balancing methodologies can be characterized as solving a trade-off between power supply and system performance. Tseng and Figueira [39] have investigated power consumption of multithreaded processes on multicore machines and found that energy-optimal configuration was usually the most efficient solution in the case of CPU-bounded tasks. Computations on several multicore nodes, communicating by MPI means, have not been investigated. Pan et al. [40] have investigated power consumption and execution time of applications from NAS parallel benchmark suite on a power-scalable cluster. However, the influence of virtualization layer has not been considered. In virtualized cloud infrastructures, server consolidation and load balancing are some

of those techniques that have gained premier importance for power-aware resource management [41]. Guo et al. [42] proposed heuristic algorithm for dynamic consolidation of heterogeneous VMs based on the analysis of the historical data.

Power models and power management algorithms are necessary for system designers to ensure that an application execution does not exceed the power constraints of a system. Moreover, performance and energy models are required for application developers to optimize time and energy consumption. The performance and energy profiles of real-life scientific applications on modern parallel architectures are not smooth or monotonous and may deviate significantly from the shapes that allowed traditional and state-of-the-art load balancing algorithms to minimize their computation time. Lastovetsky and Manumachu [43] have proposed new model-based methods and algorithms for minimization of time and energy of computations for general shapes of performance and energy profiles of data parallel applications observed on the homogeneous multicore clusters. Zhang et al. [44] introduced an analytical hierarchy process based model to perform the decision-making for virtual machine migration towards green cloud computing. The bi-objective optimization problem for performance and energy for data-parallel applications on homogeneous clusters has been formulated in [45]. Finally, the survey [46] concludes that there exists no predictive model today truly and comprehensively capturing performance and energy consumption of the highly heterogeneous and hierarchical architecture of the modern HPC node. Moreover, computational performance and energy efficiency of any non-trivial application is highly dependable on its specific features and the selection of the best suitable numerical methods [47, 48, 49].

To the best of our knowledge, there are no reports in the literature on attempts to optimize the parallel performance of real-life application on heterogeneous cloud in terms of both consumed time and energy. The novelty of the presented research is the extensive efficiency analysis, which evaluates the parallel speedup on heterogeneous cloud resources, virtualization overhead, a trade-off between the computing speed and the consumed energy, performance of the applied domain decomposition methods and application specific issues of the haemodynamic flows. The presented study supports and advances the idea that time and energy performance models need to be built as discrete functions of problem size, approximating the measured data from application tests on the particular computing architecture. Such discrete functions can be used as input for the performance and energy optimization of the considered application on the particular architecture.

3 A TARGET APPLICATION

The aortic valve has a complex 3D geometry, which is composed of three leaflets and Valsalva sinuses connected together through the commissures. The patient-specific aortic valve geometry was represented by the developed 3D geometric model [50] constructed from the parametric curves according to the obtained patient-specific geometric parameters. The 3D images of the aortic valve of a human subject were

obtained by using the computer tomography equipment GE LightSpeed VCT in the Cardiology and Angiology Centre of Vilnius University Hospital “Santaros Klinikos”. The Medical Imaging Interaction Toolkit (MITK) [51] was employed to extract the geometric parameters of the aortic valve from the obtained DICOM images (Figure 1). Finally, the geometric model constructed according to the obtained patient-specific geometric parameters was imported into the ANSYS Workbench for mesh generation.

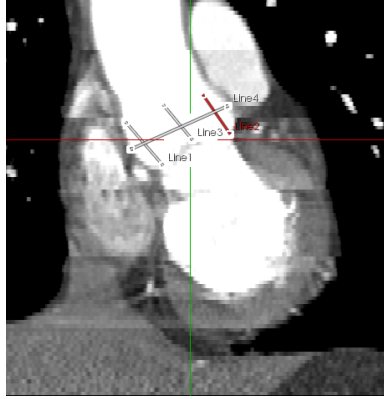


Figure 1. Extraction of the geometric parameters from DICOM images

The pulsatile flow of viscous incompressible fluid is described by the Navier-Stokes equations [52] solved by the finite volume method. The systolic phase of the cardiac cycle was simulated by applying at the inlet a time-dependent velocity of the plug flow based on the clinical Doppler measurements. The measured velocity reached the maximal value of 1.44 m/s during the phase of the peak systole $t = 0.14$ s, while the simulation time interval of 0.36 s was considered. All simulations started from a zero initial condition and the prescribed inflow was accelerated according to the measured waveform. The no-slip boundary conditions were prescribed for velocity on the aorta walls and leaflet surfaces. On the outlet, the prescribed pressure and zero velocity gradient normal to the boundary were applied. The turbulence intensity of 5 % and the hydraulic diameter equal to 0.018 m were specified on the inlet. The density of the blood was set to $\rho = 1060 \text{ kg/m}^3$. The dynamic viscosity coefficient was $\mu = 0.004028 \text{ kg/ms}$. Other details of the applied numerical model and references can be found in [53, 54].

Figure 2 shows flow vortices, illustrating the complexity of the 3D flow pattern at $t = 0.1636$ s in the aortic sinuses. Figure 2 a) presents the results obtained by using the $k - \epsilon$ turbulence model [53], while Figure 2 b) shows the data obtained by using the $k - \omega$ turbulence model [53]. The velocity field was visualized by using the streamlines coloured according to the pressure field. The developed vortices are detached and move downstream in the ascending aorta as the flow starts to decelerate after passing the phase of the peak systole. It is worth noting that different

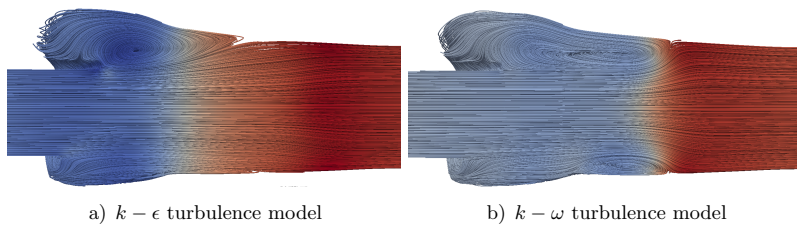


Figure 2. Flow vortices at $t = 0.1636$ s in the aortic sinuses

vertex patterns can be obtained by using different turbulence models. The previous research [54] had shown that the high Reynolds number $k - \epsilon$ turbulence model significantly smoothed the vortex field. Thus, the shear-stress transport (SST) $k - \omega$ model [53] is considered for the developed software service because it is more suitable for the simulation of relatively low Reynolds number turbulent flows past aortic valve. Moreover, it can be concluded that the considered benchmark is rather complex from the numerical point of view.

The way of coupling between the velocity and pressure is an essential part of any numerical scheme for the solution of Navier–Stokes equations [52]. There are two main strategies to perform the velocity-pressure coupling, either a segregated or a coupled approach. In the segregated approach, equations for all variables in the system are decoupled by using the fixed known values from the last iteration of the other independent variables. The linear systems obtained after the discretization can be solved separately for all variables. This approach has the advantage of yielding small storage requirements and systems amenable to solution by classical iterative methods because of the standard structure and properties of system matrices. Unfortunately, suppressed interlinkage between the partial differential equations results in serious drawbacks, such as convergence deterioration and the need for under-relaxation. The drawbacks of segregated schemes and tremendous increase in the available computer memory have stimulated the search for coupled solution algorithms [55]. The idea is to solve discretized momentum and pressure-based continuity equations together in one system of linear equations. Retaining the coupling between the momentum and pressure equations promotes the stability and accelerates the convergence rates. Application of the coupled scheme is advised when the quality of the mesh is poor, non-linear iterations are very expensive due to the time-consuming physical models for constitutive relations, or if larger time steps are required. However, for significantly reduced number of outer iterations of coupled scheme, we pay a price with a solution of four times larger systems of linear equations with non-standard matrices. There is a danger that the advantage of the higher convergence rate will be countered by the increase in computational time incurred in the solution of the enlarged system of equations. Thus, the coupled [55, 5] and PISO [56] schemes were considered as the alternative algorithms to investigate the performance of the developed software service.

Simulations of the turbulent aortic valve flows are time consuming, therefore, parallel computations have become an obvious option for significantly reducing computing time. Domain decomposition approach and message passing technology were used for parallel computations performed by ANSYS Fluent. The default option of MPI library employed by ANSYS Fluent was IBM Platform MPI 9.1.4.2. Communication pattern was highly influenced by the applied domain decomposition method because each process, working on its subdomain, mostly communicates with processes, working on neighbouring subdomains. The most often the domain decompositions were performed by ParMETIS library. However, the alternative Cartesian axes and cylindrical z-coordinate methods were also considered. Solving the largest benchmark problem of 3.2 million cells on 5 nodes (20 cores) and performing the domain decomposition by ParMETIS, 227 MB and 405 MB were transferred during one iteration in the case of the coupled and PISO solution algorithms, respectively. 55 388 MB and 58 320 MB were transferred during the whole test run in the case of the coupled and PISO schemes, respectively. The peak memory consumption of the largest benchmark problem solved on one node reached 11.5 and 9.0 GB, in the case of the coupled and PISO schemes, respectively. In the case of the largest benchmark problem of 3.2 million finite volumes solved on 5 nodes (20 cores), the averaged CPU utilization was equal to 99.84 % and 99.25 % for the coupled and PISO solution algorithms, respectively. Thus, the initial investigation showed that performance of haemodynamic computations might depend on the numerical solution algorithm, the number of used nodes (cores) and the applied domain decomposition method.

4 CLOUD INFRASTRUCTURE AND THE DEVELOPED SOFTWARE SERVICES

The university private cloud infrastructure based on OpenStack Stein version [11] is hosted in Vilnius Gediminas Technical University. The deployed capabilities of the OpenStack cloud infrastructure include Compute Service Nova, Networking Service Neutron, Image Service Glance, Identity Service Keystone, Object Storage Service Swift and Block Storage Service Cinder. The Ubuntu 18.04.3 LTS release was installed in the host nodes. Linux containers were managed with Docker 19.03.2, which created an abstraction layer between computing resources and the services using them. The containers had the following characteristics: 4 cores, 31.2 GB RAM, 80 GB HDD and Ubuntu 18.04.3 LTS release. The cloud infrastructure is composed of several different types of nodes connected to 1 Gbps Ethernet LAN. Hardware characteristics of faster nodes hosting the containers are listed below: Intel® Core i7-6700 3.40 GHz CPU (4 cores), 32 GB DDR4 2133 MHz RAM and 1 TB HDD. Hardware characteristics of slower nodes are listed below: Intel Core i7-4790 3.60 GHz CPU (4 cores), 32 GB DDR3 1866 MHz RAM and 1 TB HDD.

The OpenStack cloud IaaS provides the platforms to develop and deploy software services called SaaS (Figure 3). In the developed cloud infrastructure, only

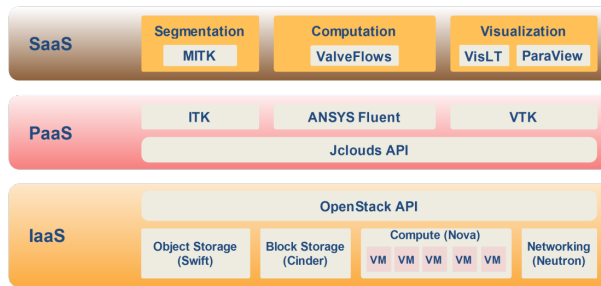


Figure 3. The layers of cloud services

one virtualization layer is used to limit virtualization overhead. In the considered implementation, OpenStack executes and manages Docker containers on bare metal nodes. The cloud infrastructure is managed by using jclouds API [57] providing unified access to EC2 services. The jclouds API binds HTTP/REST services to synchronous and asynchronous Java APIs. Moreover, jclouds API supports 30 cloud providers and cloud software stacks offering high portability. A user-friendly Cloud Manager [26] has been developed by using jclouds. Monitoring of the running instances, volumes and images is available for the user. SaaS jobs can be run by the developed launchers and monitored by the Cloud Manager. The platform as a service was provided on the basis of the popular numerical modelling software ANSYS Fluent [53], which was widely used by researchers and engineers to solve various CFD applications. ITK [58] was deployed on cloud infrastructure as PaaS for developing medical image processing applications. ITK is a cross-platform, open-source application development framework widely used for the development of image segmentation and image registration software. ITK employs leading-edge algorithms for registering and segmenting multidimensional data. A lot of software tools and environments have been developed by using ITK to segment structures in 3D medical images. A Visualization Toolkit (VTK) [59] is deployed as the platform for developing visualization software. The academic numerical software developed by the university researchers usually lacks the required visualization capabilities, therefore, a wide variety of visualization algorithms provided by VTK can fill this gap in the cloud infrastructure. The applications of the discussed toolkits are platform independent, which is very attractive for heterogeneous cloud architectures.

The SaaS layer contains software services developed on top of the provided platforms (Figure 3). Software services were provided for performing medical image segmentation to obtain the patient-specific geometry and to prepare the patient-specific model of the aortic valve. The medical image segmentation was performed and geometric parameters were obtained by using the MITK [51], which was based on the ITK platform, but also used VTK. The software service ValveFlows was developed by using ANSYS Fluent for computations of the patient-specific aortic

valves [26]. Computational results were visualized using the open-source ParaView software [60] and the cloud visualization service VisLT [61] deployed on top of the VTK platform. VisLT was supplemented with the developed middleware component, which could reduce the communication between different parts of the cloud infrastructure.

5 THE ANALYSIS OF PARALLEL PERFORMANCE AND ENERGY CONSUMPTION

The presented analysis aims at investigating the parallel performance and energy consumption of the developed software services for haemodynamic flow computations on Docker containers managed by the OpenStack cloud infrastructure. Initially, the considered benchmarks were solved on 5 nodes with 4 cores each, which resulted in homogeneous virtual architecture of 5 Docker containers using 20 cores. 3 slower nodes with 4 cores each were added to perform computational experiments on heterogeneous virtual resources, which resulted in 8 containers using 32 cores. The following subsections present the results of the performed research on virtualization overhead, parallel performance, the influence of the applied domain decomposition methods, power consumption, energy-efficiency and a trade-off between the computing time and the consumed energy.

5.1 Execution Time and Virtualization Overhead

First, the computational performance of the considered numerical algorithms and overhead induced by the virtualization were investigated. The benchmark problem was solved on 8 heterogeneous containers (8 nodes, 32 cores) using discrete meshes of the increasing size of 0.8, 1.6 and 3.2 million cells. In Figure 4a), execution times obtained solving the benchmark on the OpenStack cloud are presented. In accordance with our previous findings [28], the solution times obtained by using the PISO numerical algorithm (P08, P16, P32) were shorter than those attained by using the coupled numerical algorithm (C08, C16, C32).

In the present research, the size of virtualization overhead was also investigated. In Figure 4b), the percentage difference in performance between the native hardware and the Docker containers is presented. It can be observed that the relative time difference is smaller than 1% for tests on one and two nodes, i.e. using 1, 4 and 8 processes. These findings are consistent with the results reported in the literature [23, 24, 26] and confirm the low overhead of the employed Docker containers. However, the growth of the virtualization overhead can be observed, when the number of nodes for the solution of the fixed size problem is increased. It is worth noting that the overhead obtained by using the coupled numerical algorithm is consistently bigger. These effects were caused by the increasing part of the communication time in the overall solution time. For the increasing number of parallel processes, the latency of the network communication becomes more and more important, especially,

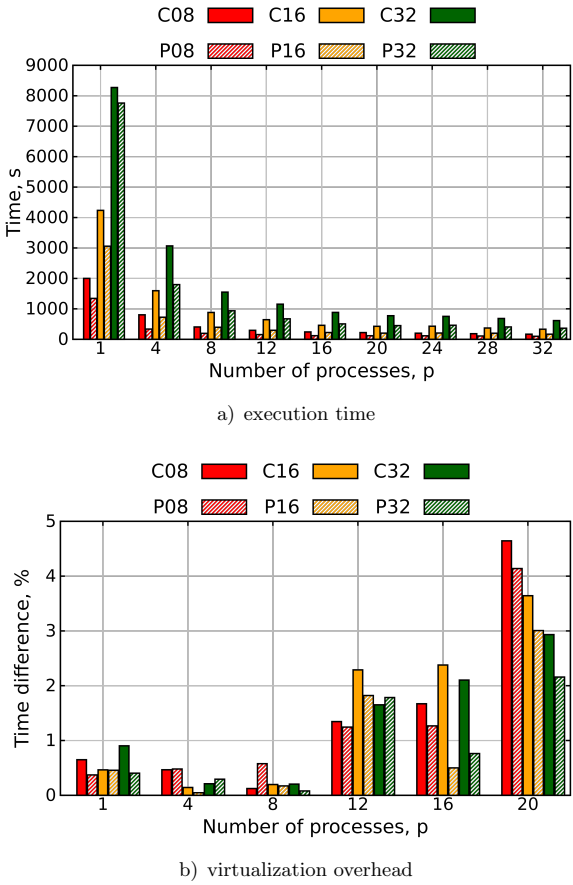


Figure 4. The performance of the developed cloud software service

for smaller size problems. On average, the measured latency of the native network was $17.5 \mu s$, while that of the virtual network was $23.0 \mu s$, which revealed 24 % latency increase. The increased latency of a virtual network has also been reported in the literature [22].

5.2 Parallel Performance

At the next stage, the parallel scalability and efficiency of the considered algorithms were studied by performing computations on heterogeneous cloud infrastructure.

In Figure 5, the parallel scalability results obtained on 5 faster homogeneous nodes (20 cores totally) are presented. The parallel scalability is evaluated by using parallel speedup values $S_p = T_1/T_p$, where T_1 and T_p are execution times measured

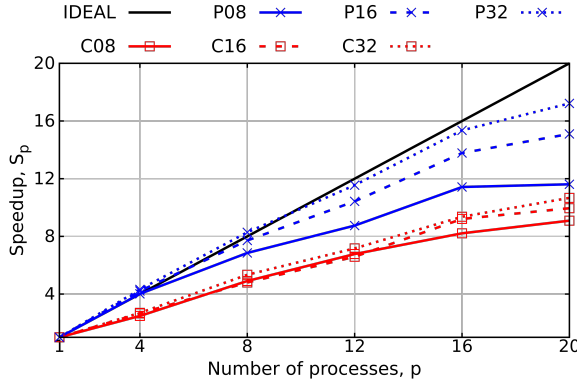
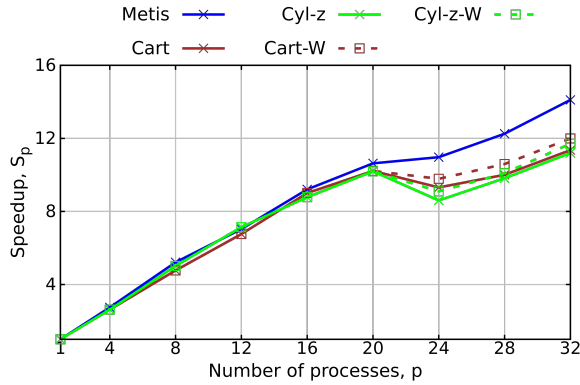


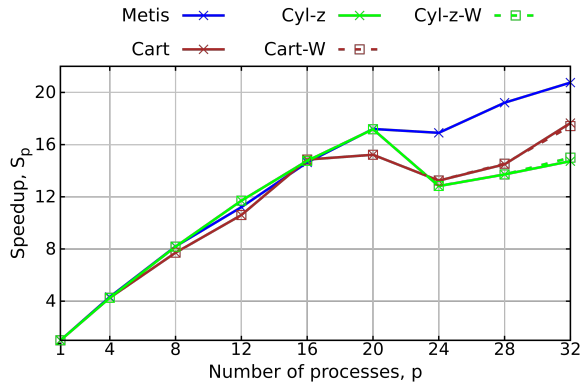
Figure 5. Speedup of parallel computation on homogeneous resources

by solving the benchmark problem with 1 and p processes, respectively. It can be observed that the parallel PISO algorithm demonstrated significantly higher parallel scalability and efficiency than the parallel coupled algorithm. As could be expected, the parallel performance metrics (speedup and efficiency) were greater for a bigger problem. This is in accordance with the theory of parallel algorithms. Noticeably, the difference is more significant for the PISO algorithm, as the performance of the coupled algorithm is bounded by the capacity of the network even for the problems of a larger size. The parallel PISO algorithm shows even super-linear speedup due to the cache effects associated with smaller working data sets for a fixed size problem and the increasing number of the employed cores. It is interesting to note that, in this study, the measured parallel performance metrics are much better than those obtained in the previous research on OpenFOAM-based parallel solvers [47, 62].

In the present research, parallel computation tests were performed using virtualized heterogeneous resources by adding up to 3 slower nodes. The performance of the slower nodes was 12% and 20% lower according to Linpack and ValveFlows benchmarks, respectively. Figure 6 presents the parallel speedup values measured using heterogeneous cloud resources (8 nodes, 32 cores). The sequential time measured on one faster core is considered for calculation of speedup values. The largest problem with 3.2 million cells was solved by using the coupled (Figure 6a)) and PISO (Figure 6b)) algorithms. It is well-known that the parallel performance of this type of numerical algorithms largely depends on the quality of the domain decomposition. The domain decomposition method should not only ensure the load balance between the parallel processes, but also minimize the amount of communications between the neighbouring regions. In this work, the performance of three domain decomposition methods was investigated. The applied method from the well-known ParMETIS library (the curve “Metis” in Figure 6) is based on the parallel multilevel multiconstraint k -way graph partitioning [63]. The Cartesian axes method (the curve “Cart” in Figure 6) uses the bisection of the domain per-



a) The coupled algorithm



b) The PISO algorithm

Figure 6. Speedup of parallel computation on heterogeneous resources

pendicular to all coordinate axes [53]. The cylindrical z -coordinate method (the curve “Cyl- z ” in Figure 6) bisects the domain only along the z cylindrical coordinate [53].

It is worth noting that the performance results of all three domain decomposition methods were quite similar up to 20 cores (5 homogeneous nodes). However, the addition of the first slower node was not beneficial. On the contrary, it caused the decrease in the speedup, i.e. the increase in the solution time, which was significant in most of the cases. This observation once more illustrated the critical dependence of considered parallel application on the network performance. It also should be noted that slower nodes are not only computationally slower, but have 42.5% higher virtual network latency ($40.0\ \mu\text{s}$ instead of $23.0\ \mu\text{s}$) as well. However, overall performance increase can be achieved by adding 2 or 3 slower nodes. In

this case, the heterogeneous nodes made 37.5 % of all resources, and a considerable increase in speedup values was achieved.

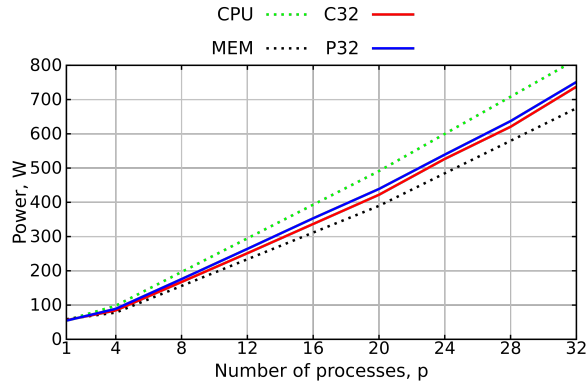
The heterogeneous load was evaluated, assigning the relative weights to the slower cores in the cases of using the Cartesian axes and the cylindrical z -coordinate methods. The parallel performance was improved, but the observed increase in the speedup was not significant (the curves “Cart-W” and “Cyl-z-W” in Figure 6). Parallel computations, using the domain decomposition performed by Metis, revealed higher speedup values. Thus, the most important factor was the ability of the multilevel graph partitioning method to reduce the amount of communication between the neighbouring regions in the case of a larger number of processes, the increased data transfer and higher network latency.

5.3 Power Consumption

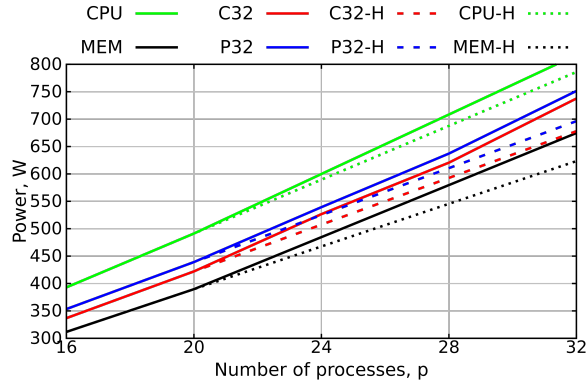
Figure 7 presents the averaged values of the consumed power in watts measured by performing a synthetic CPU benchmark (the curve “CPU”), a synthetic memory benchmark (the curve “MEM”), the computations based on the coupled algorithm (the curve “C32”) and the computations based on the PISO algorithm (the curve “P32”). The power was measured by using the EnerGenie energy meter EGM-PWM-LAN. The representative computational experiments were performed 10 times, and the standard deviations were computed. The results obtained by solving the benchmark problem with 3.2 million cells on 8 heterogeneous nodes (32 cores totally) are shown in Figure 7. In agreement with our previous findings, power measurements demonstrated that the PISO algorithm was better in utilizing the CPU cores, i.e., it was more CPU-intensive than the coupled algorithm. Consequently, the software service ValveFlows based on the PISO algorithm consumed more power per fixed time interval, but solved the considered benchmark faster (Figure 4 a)). The power difference became noticeable, when 8 processes (2 nodes) were employed. However, it stabilized at around 20 W on average and did not grow further, increasing the number of the employed nodes, as could be expected from the growing differences in computational performance (Figures 5 and 6).

In Figure 7 b), the effects of heterogeneity are highlighted by showing the extrapolated power consumption levels for homogeneous hardware (the curves “CPU-H”, “MEM-H”, “C32-H” and “P32-H”). These results show that slower nodes are also less energy-efficient, which is in agreement with the values of thermal design power provided by the CPU manufacturer. All these findings raise the question, whether the observed increase in power consumption for the PISO algorithm and heterogeneous setup is compensated by the obtained computational performance gains, i.e. the reduction in the execution time. Such questions will be addressed in terms of energy-efficiency in the next section.

Another important problem associated with the influence of virtualization on the changes in power consumption was also addressed in the present research. Figure 8 shows the relative difference in power consumption between the native hardware and the OpenStack cloud, running the considered software based on the coupled



a) The global view



b) The zoomed view with highlighted effects of heterogeneity

Figure 7. Power consumption

algorithm. Up to 0.6% difference could be observed in the results of the performed benchmark. It is worth noting that the differences declined, when the number of employed cores was increased. As can be seen from Figure 7 a), utilization of cores decreased in comparison with the CPU benchmark and, consequently, the influence of the virtualization on power consumption also declined. The standard deviation was equal to 0.11% and 0.08% of the presented averaged values in the case of 1 node (4 cores) and 8 nodes (32 cores), respectively. Thus, the standard deviation was smaller than the observed difference in power consumption between the native hardware and OpenStack cloud in all considered cases.

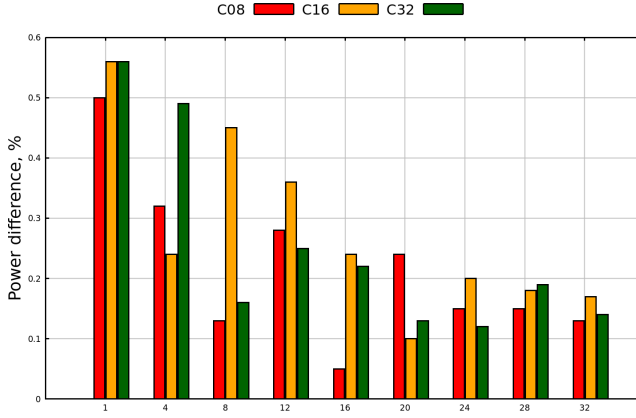


Figure 8. The percentage difference in power consumption between the native hardware and the OpenStack cloud

5.4 Energy-Efficiency

Using the results given in the previous sections, the best solution algorithm and hardware setup in terms of the overall energy consumption can be found. The consumed energy E , required to solve the considered problem, was computed as

$$E = P \cdot T_{exec} \quad (1)$$

where P is the average power and T_{exec} is the solution time. The consecutive energy change ΔE , increasing the number of the employed cores ($p_i = 1, 4, \dots, 32$), was calculated by the formula

$$\Delta E = E_{p_{i-1}} - E_{p_i} \quad (2)$$

where $E_{p_{i-1}}$ is the energy required for the solution of the considered problem on p_i cores. The values of the consumed energy (1) are shown in Figure 9 a). It can be observed that the parallel PISO algorithm consumed significantly less energy than the parallel coupled algorithm. Thus, the PISO algorithm is superior in terms of computational performance and energy-efficiency because less energy is required in spite of a slightly higher instant power consumption.

As concerns the best hardware setup, the situation is more complicated. Figure 9 b) presents the values of the consecutive energy change calculated by the formula (2). In terms of energy-efficiency, the most significant consecutive change in the consumed energy occurs, when all 4 cores of a single node are used. The observed energy reduction is equal to 45.1 % and 62.3 %, in the case of the largest problem with 3.2 million cells, solved by using the coupled algorithm and the PISO algorithm, respectively. This finding is in agreement with the current trends in the design of modern processors, which is motivated by higher energy-efficiency. Further

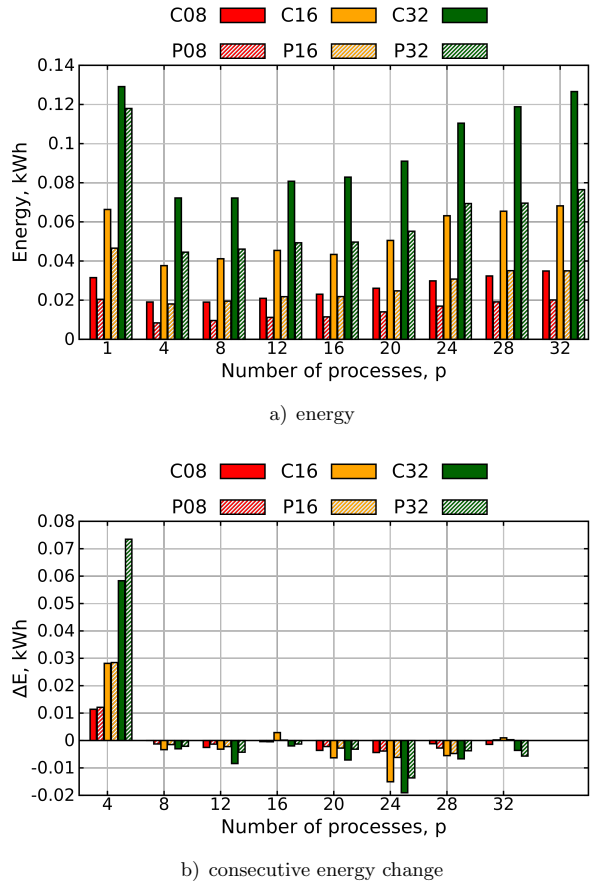


Figure 9. Energy consumption

analysis of the presented results reveals a significant jump in energy consumption for the heterogeneous setup with one slower node, which is caused by the above-discussed performance degradation (Figure 6). However, in most of the cases, the increase in energy consumption for 8, 12, 16, and even 20 processes, is not significant. The question arises, whether it is rational to choose the optimal hardware setup as a case of minimal energy consumption, when it is known that the additional nodes cause a significant reduction in the solution time. To answer this question, a bi-objective optimization problem needs to be considered.

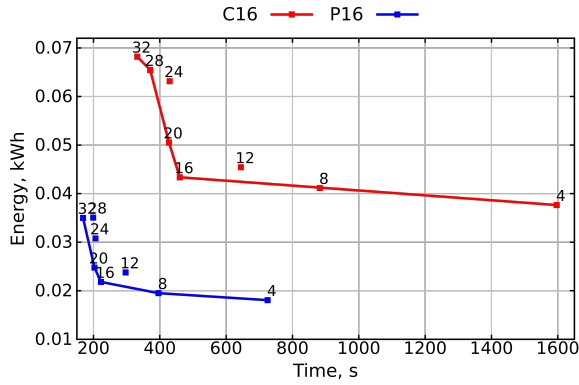
5.5 The Solution of a Bi-Objective Optimization Problem

The choice of the optimal hardware setup needs to be taken in the presence of two conflicting objectives or criteria: the solution time T and the consumed energy E .

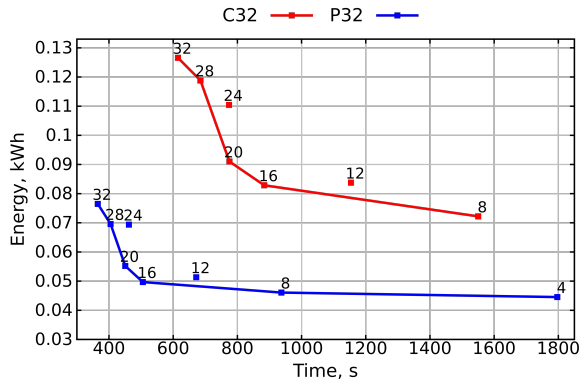
This bi-objective optimization problem can be formulated as follows:

$$\min_{p_i \in X} (T(p_i), E(p_i)) \quad (3)$$

where $X = \{1, 4, 8, 16, 20, 24, 28, 32\}$ is the set of feasible solutions. There are many different approaches to deal with multi-objective optimization problems. For non-trivial problems, no single solution exists, which simultaneously minimizes each objective. A common approach is to find the Pareto optimal solutions, i.e., the solutions that cannot be improved in any of the objectives without degrading at least one of the objectives. The set of the Pareto optimal solutions is often called the Pareto front. For the formulated bi-objective optimization problem (3), the Pareto optimal solutions can be found from the scalar plot shown in Figure 10.



a) the problem with 1.6 million cells



b) the problem with 3.2 million cells

Figure 10. Pareto fronts, considering energy and computing time as objectives

Figure 10 demonstrates a clear domination of the parallel PISO algorithm over the parallel coupled algorithm. The hardware setup with just one slower node ($p_i = 24$) is not the Pareto optimal in all presented cases. This result can be expected from the previous analysis. It is worth noting that the homogeneous setup with three fast nodes ($p_i = 12$) is not the Pareto optimal in all four cases either. This finding can be explained by a relatively less successful domain decomposition produced by Metis for this number of processes. In terms of Pareto optimality, all other hardware configurations cannot be excluded in all the considered cases either. Other approaches, including subjective preferences of a decision maker, should be considered to find a single solution to the formulated problem. Scalarization can be considered as a popular approach to solve a multi-objective optimization problem. The idea is to convert the original problem with multiple objectives to a single-objective optimization problem, which is referred to as a scalarized problem. A proper scalarization method ensures the Pareto optimality of the obtained solutions. In the case of the considered bi-objective optimization problem, linear scalarization can be defined as follows:

$$\min_{p_i \in X} (\omega_T \hat{T}(p_i) + \omega_E \hat{E}(p_i)) \quad (4)$$

where ω_T and ω_E are the weights of the normalized solution time objective $\hat{T}(p_i)$ and the normalized consumed energy objective $\hat{E}(p_i)$, respectively. The parameters of the scalarization, ω_T and ω_E , are set by a decision maker, but should satisfy simple conditions: $\omega_T + \omega_E = 1$, $1 \geq \omega_T \geq 0$ and $1 \geq \omega_E \geq 0$.

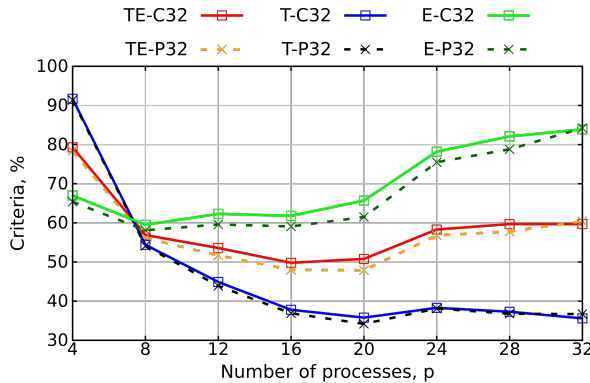


Figure 11. The application of linear scalarization method with various weights

The results of linear scalarization of the considered bi-objective optimization problem (4) are shown in Figure 11. In the case of the largest problem with 3.2 million cells solved by using the coupled algorithm, the curves TE-C32, T-C32 and E-C32 represent the objective functions, with equal, solution time- and energy-oriented weights, respectively. Other curves, TE-P32, T-P32 and E-P32, represent

the objective functions of the same problem solved by the PISO algorithm. Thus, three representative sets of weights were considered in the work. The equal weights ($\omega_T = \omega_E = 0.5$) resulted in optimal configurations based on 16 and 20 cores for the coupled algorithm and the PISO algorithm, respectively. The energy-oriented weights ($\omega_T = 0.2, \omega_E = 0.8$) gave the optimal hardware configuration based on 8 cores for both numerical algorithms. On the contrary, the solution time-oriented weights ($\omega_T = 0.8, \omega_E = 0.2$) revealed the optimal configurations based on 20 homogeneous and 32 heterogeneous cores for the PISO algorithm and the coupled algorithm, respectively. Various optimal hardware configurations obtained for different numerical algorithms proved that the considered bi-objective optimization problem was not trivial even for a simple set of hardware configurations and revealed some challenges for decision-makers.

6 CONCLUSIONS

In this article, parallel performance and energy consumption analysis of the haemodynamic computations performed using Docker containers of the heterogeneous OpenStack cloud infrastructure is presented. Based on the performed investigation, some observations and concluding remarks may be drawn as follows:

- Virtualization layer reduced computational performance of the developed software services by less than 1 % in the case of one or two nodes used. Increasing the number of the employed nodes caused an increase in the virtualization overhead to 4.6 % of the benchmark time on the native hardware due to higher latency of the virtual network.
- The employed virtualization has not significant influence to power consumption of the performed computations. The largest measured difference was less than 0.6 % of the power consumed by running the benchmark on the native hardware.
- The parallel PISO algorithm demonstrated significantly higher computational performance and better parallel scalability than the parallel coupled algorithm for the computations of the considered haemodynamic flows.
- Power measurements demonstrated that the PISO algorithm was more CPU intensive and consumed more power per fixed time interval than the coupled algorithm. However, the consumed energy analysis revealed that the PISO algorithm required less energy to solve the considered problems due to higher computational performance.
- The minimal amount of energy is required to solve the considered problems using a single node with all cores employed. Energy consumption slightly increased with the increasing number of the employed nodes until the degradation of parallel performance became significant. The largest increase in the consumed energy could be observed, when the first slower node was employed.

- Bi-objective optimization based on Pareto front analysis or using the linear scalarization method could help to solve a trade-off between the computing time and the consumed energy.
- The Pareto front analysis helped to detect inefficient hardware configurations. The heterogeneous setup with a single slower node and the homogeneous setup with less successful domain decomposition to 12 parts were not the Pareto optimal in all the considered cases.
- In the case of the considered application, linear scalarization with weights suggested completely different hardware configurations for various subjective preferences of a decision-maker.
- The conducted study demonstrated great challenges to the efficient use of the heterogeneous cloud resources by the developed software service in the case of the considered application. Optimal hardware configurations for single parallel jobs were highly dependent on the network properties, the numerical algorithm and the results of the applied domain decomposition method.
- The performed research has revealed that standard benchmarks can hardly provide comprehensive information required for time- and energy-efficient scheduling of parallel haemodynamic computations. The preliminary specific benchmarks are required to evaluate the parallel performance of the developed software services and algorithmic aspects of the considered application.

REFERENCES

- [1] MENDIS, S.—PUSKA, P.—NORRVING, B. (Eds.): *Global Atlas on Cardiovascular Disease Prevention and Control: Policies, Strategies and Interventions*. World Health Organization, World Heart Federation, World Stroke Organization, Geneva, 2011.
- [2] MOOSAVI, M.-H.—FATOURAEE, N.—KATOOZIAN, H.—PASHAEI, A.—CAMARA, O.—FRANGI, A. F.: Numerical Simulation of Blood Flow in the Left Ventricle and Aortic Sinus Using Magnetic Resonance Imaging and Computational Fluid Dynamics. *Computer Methods in Biomechanics and Biomedical Engineering*, Vol. 17, 2014, No. 7, pp. 740–749, doi: 10.1080/10255842.2012.715638.
- [3] MAROM, G.: Numerical Methods for Fluid-Structure Interaction Models of Aortic Valves. *Archives of Computational Methods in Engineering*, Vol. 22, 2015, No. 4, pp. 595–620, doi: 10.1007/s11831-014-9133-9.
- [4] TUMONIS, L.—KAČIANAUSKAS, R.—KAČENIAUSKAS, A.—SCHNEIDER, M.: The Transient Behavior of Rails Used in Electromagnetic Railguns: Numerical Investigations at Constant Loading Velocities. *Journal of Vibroengineering*, Vol. 9, 2007, No. 3, pp. 15–19.
- [5] KAČENIAUSKAS, A.—RUTSCHMANN, P.: Parallel FEM Software for CFD Problems. *Informatica*, Vol. 15, 2004, No. 3, pp. 363–378, doi: 10.15388/Informatica.2004.066.

- [6] CHNAFA, C.—MENDEZ, S.—NICOUD, F.—MORENO, R.—NOTTIN, S.—SCHUSTER, I.: Image-Based Patient-Specific Simulation: A Computational Modelling of the Human Left Heart Haemodynamics. *Computer Methods in Biomechanics and Biomedical Engineering*, Vol. 15, 2012, No. 1, pp. 74–75, doi: 10.1080/10255842.2012.713673.
- [7] BASTRAKOV, S.—MEYEROV, I.—GERGEL, V.—GONOSKOV, A.—GORSHKOV, A.—EFIMENKO, E.—IVANCHENKO, M.—KIRILLIN, M.—MALOVA, A.—OSIPOV, G.—PETROV, V.—SURMIN, I.—VILDEMANOV, A.: High Performance Computing in Biomedical Applications. *Procedia Computer Science*, Vol. 18, 2013, pp. 10–19, doi: 10.1016/j.procs.2013.05.164.
- [8] SAKELLARI, G.—LOUKAS, G.: A Survey of Mathematical Models, Simulation Approaches and Testbeds Used for Research in Cloud Computing. *Simulation Modelling Practice and Theory*, Vol. 39, 2013, pp. 92–103, doi: 10.1016/j.simpat.2013.04.002.
- [9] SEHDEV, G. K.—KUMAR, A.: Performance Evaluation of Power Aware VM Consolidation Using Live Migration. *International Journal of Computer Network and Information Security*, Vol. 7, 2015, No. 2, pp. 67–76, doi: 10.5815/ijcnis.2015.02.08.
- [10] ZAKARYA, M.—GILLAM, L.: Energy Efficient Computing, Clusters, Grids and Clouds: A Taxonomy and Survey. *Sustainable Computing: Informatics and Systems*, Vol. 14, 2017, pp. 13–33, doi: 10.1016/j.suscom.2017.03.002.
- [11] OpenStack. 2019, available at: <https://www.openstack.org>.
- [12] NURMI, D.—WOLSKI, R.—GRZEGORCZYK, C.—OBTELLI, G.—SOMAN, S.—YOUSSEF, L.—ZAGORODNOV, D.: The Eucalyptus Open-Source Cloud-Computing System. *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'09)*, Shanghai, China, 2009, pp. 124–131, doi: 10.1109/CCGRID.2009.93.
- [13] CHIERICI, A.—VERALDI, R.: A Quantitative Comparison Between XEN and KVM. *Journal of Physics: Conference Series*, Vol. 219, 2010, No. 4, Art.No. 042005, pp. 1–10, doi: 10.1088/1742-6596/219/4/042005.
- [14] LXC. 2019, available at: <https://linuxcontainers.org>.
- [15] Docker. 2019, available at: <https://www.docker.com>.
- [16] McMILLAN, B.—CHEN, C.: High Performance Docking. Technical Report, 2014.
- [17] PRIEDHORSKY, R.—RANDLES, T.: Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'17)*, ACM, 2017, Art. No. 36, pp. 1–10, doi: 10.1145/3126908.3126925.
- [18] KURTZER, G. M.—SOCHAT, V.—BAUER, M. W.: Singularity: Scientific Containers for Mobility of Compute. *PLoS ONE*, Vol. 12, 2017, No. 5, Art.No. e0177459, pp. 1–20, doi: 10.1371/journal.pone.0177459.
- [19] SAUVANAUD, C.—DHOLAKIA, A.—GUITART, J.—KIM, C.—MAYES, P.: Big Data Deployment in Containerized Infrastructures Through the Interconnection of Network Namespaces. *Software: Practice and Experience*, Vol. 50, 2020, No. 7, pp. 1087–1113, doi: 10.1002/spe.2793.
- [20] ANSYS 18.0 Fluids and Structures. UberCloud, 2019, available at: <https://www.theubercloud.com/ansys-cloud>.

- [21] SEO, K.-T.—HWANG, H.-S.—MOON, I.-Y.—KWON, O.-Y.—KIM, B.-J.: Performance Comparison Analysis of Linux Container and Virtual Machine for Building Cloud. *Advanced Science and Technology Letters*, Vol. 66, 2014, pp. 105–111.
- [22] KAČENIAUSKAS, A.—PACEVIČ, R.—STAŠKŪNIENĖ, M.—ŠEŠOK, D.—RUSAKEVIČIUS, D.—AIDIETIS, A.—DAVIDAVIČIUS, G.: Private Cloud Infrastructure for Applications of Mechanical and Medical Engineering. *Information Technology and Control*, Vol. 44, 2015, No. 3, pp. 254–261, doi: 10.5755/j01.itc.44.3.7379.
- [23] DI TOMMASO, P.—PALUMBO, E.—CHATZOU, M.—PRIETO, P.—HEUER, M. L.—NOTREDAME, C.: The Impact of Docker Containers on the Performance of Genomic Pipelines. *PeerJ*, Vol. 3, 2015, Art. No. e1273, doi: 10.7717/peerj.1273.
- [24] MAZZONI, E.—AREZZINI, S.—BOCCALI, T.—CIAMPA, A.—COSCETTI, S.—BONACORSI, D.: Docker Experience at INFN-Pisa Grid Data Center. *Journal of Physics: Conference Series*, Vol. 664, 2015, No. 2, Art. No. 022029, pp. 22–29, doi: 10.1088/1742-6596/664/2/022029.
- [25] ESTRADA, Z. J.—DENG, F.—STEPHENS, Z.—PHAM, C.—KALBARCZYK, Z.—IYER, R.: Performance Comparison and Tuning of Virtual Machines for Sequence Alignment Software. *Scalable Computing: Practice and Experience*, Vol. 16, 2015, No. 1, pp. 71–84, doi: 10.12694/scpe.v16i1.1061.
- [26] KAČENIAUSKAS, A.—PACEVIČ, R.—STARIKOVIČIUS, V.—MAKNICKAS, A.—STAŠKŪNIENĖ, M.—DAVIDAVIČIUS, G.: Development of Cloud Services for Patient-Specific Simulations of Blood Flows Through Aortic Valves. *Advances in Engineering Software*, Vol. 103, 2017, pp. 57–64, doi: 10.1016/j.advengsoft.2016.01.013.
- [27] HAN, J.—AHN, J.—KIM, C.—KWON, Y.—CHOI, Y.—HUH, J.: The Effect of Multi-Core on HPC Applications in Virtualized Systems. In: Guarracino, M. R. et al. (Eds.): *Euro-Par 2010 Parallel Processing Workshops (Euro-Par 2010)*. Springer, Berlin, Heidelberg, *Lecture Notes in Computer Science*, Vol. 6586, 2011, pp. 615–623, doi: 10.1007/978-3-642-21878-1_76.
- [28] STARIKOVIČIUS, V.—KAČENIAUSKAS, A.—MAKNICKAS, A.—STUPAK, E.—PACEVIČ, R.—STAŠKŪNIENĖ, M.—DAVIDAVIČIUS, G.: On Efficiency of Parallel Solvers for the Blood Flow Through Aortic Valve. *Mathematical Modelling and Analysis*, Vol. 22, 2017, No. 5, pp. 601–616, doi: 10.3846/13926292.2017.1339642.
- [29] XAVIER, M. G.—NEVES, M. V.—ROSSI, F. D.—FERRETO, T. C.—LANGE, T.—DE ROSE, C. A. F.: Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, IEEE*, 2013, pp. 233–240, doi: 10.1109/PDP.2013.41.
- [30] HALE, J.—LI, L.—RICHARDSON, C. N.—WELLS, G. N.: Containers for Portable, Productive and Performant Scientific Computing. *Computing in Science and Engineering*, Vol. 19, 2017, No. 6, pp. 40–50, doi: 10.1109/MCSE.2017.2421459.
- [31] MOHAMMADI, M.—BAZHIROV, T.: Comparative Benchmarking of Cloud Computing Vendors with High Performance Linpack. *Proceedings of the 2nd International Conference on High Performance Compilation, Computing and Communications (HP3C)*, 2018, pp. 1–5, doi: 10.1145/3195612.3195613.

- [32] STAŠKŪNIENĖ, M.—KAČENIAUSKAS, A.—STARIKOVIČIUS, V.—MAKNICKAS, A.—STUPAK, E.—PACEVIČ, R.: Parallel Simulation of the Aortic Valve Flows on the OpenStack Cloud. *Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, 2017, Art.No. 16, doi: 10.4203/ccp.111.16.
- [33] VALENTINI, G. L.—LASSONDE, W.—KHAN, S. U.—MIN-ALLAH, N.—MADANI, S. A.—LI, J.—ZHANG, L.—WANG, L.—GHANI, N.—KOŁODZIEJ, J.—LI, H.—ZOMAYA, A. Y.—XU, C.-Z.—BALAJI, P.—VISHNU, A.—PINEL, F.—PECERO, J. E.—KLIÁZOVICH, D.—BOUVRY, P.: An Overview of Energy Efficiency Techniques in Cluster Computing Systems. *Cluster Computing*, Vol. 16, 2013, No. 1, pp. 3–15, doi: 10.1007/s10586-011-0171-x.
- [34] ALBERS, S.: Energy-Efficient Algorithms. *Communications of the ACM*, Vol. 53, 2010, No. 5, pp. 86–96, doi: 10.1145/1735223.1735245.
- [35] MISHRA, A.—KHARE, N.: Analysis of DVFS Techniques for Improving the GPU Energy Efficiency. *Open Journal of Energy Efficiency*, Vol. 4, 2015, No. 4, pp. 77–86, doi: 10.4236/ojee.2015.44009.
- [36] AL HASIB, A.—NATVIG, L.—KJELDSBERG, P.—CEBRIÁN, J.: Energy Efficiency Effects of Vectorization in Data Reuse Transformations for Many-Core Processors. *Journal of Low Power Electronics and Applications*, Vol. 7, 2017, No. 1, Art.No. 5, 21 pp., doi: 10.3390/jlpea7010005.
- [37] BAUN, C.: Performance and Energy-Efficiency Aspects of Clusters of Single Board Computers. *International Journal of Distributed and Parallel Systems*, Vol. 7, 2016, No. 2-4, pp. 13–22, doi: 10.5121/ijdps.2016.7402.
- [38] PINHEIRO, E.—BIANCHINI, R.—CARRERA, E. V.—HEATH, T.: Dynamic Cluster Reconfiguration for Power and Performance. In: Benini, L., Kandemir, M., Ramanujam, J. (Eds.): *Compilers and Operating Systems for Low Power*. Springer, Boston, MA, 2003, pp. 75–93, doi: 10.1007/978-1-4419-9292-5.5.
- [39] TSENG, C.—FIGUEIRA, S.: An Analysis of the Energy Efficiency of Multi-Threading on Multi-Core Machines. *International Conference on Green Computing*, IEEE, 2010, pp. 283–290, doi: 10.1109/GREENCOMP.2010.5598301.
- [40] PAN, F.—FREEH, V. W.—SMITH, D. M.: Exploring the Energy-Time Tradeoff in High-Performance Computing. *19th IEEE International Parallel and Distributed Processing Symposium*, 2005, pp. 1–9, doi: 10.1109/IPDPS.2005.213.
- [41] CHOUDHARY, A.—RANA, S.—MATAHAI, K. J.: A Critical Analysis of Energy Efficient Virtual Machine Placement Techniques and Its Optimization in a Cloud Computing Environment. *Procedia Computer Science*, Vol. 78, 2016, pp. 132–138, doi: 10.1016/j.procs.2016.02.022.
- [42] GUO, L.—ZHANG, Y.—ZHAO, S.: Heuristic Algorithms for Energy and Performance Dynamic Optimization in Cloud Computing. *Computing and Informatics*, Vol. 36, 2017, No. 6, pp. 1335–1360, doi: 10.4149/cai.2017.6.1335.
- [43] LASTOVETSKY, A.—MANUMACHU, R. R.: New Model-Based Methods and Algorithms for Performance and Energy Optimization of Data Parallel Applications on Homogeneous Multicore Clusters. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 28, 2017, No. 4, pp. 1119–1133, doi: 10.1109/TPDS.2016.2608824.

- [44] ZHANG, L.—MA, J.—LIU, T.—WANG, Y.—LU, D.: AHP Aided Decision-Making in Virtual Machine Migration for Green Cloud. *Computing and Informatics*, Vol. 37, 2018, No. 2, pp. 291–310, doi: 10.4149/cai.2018.2.291.
- [45] MANUMACHU, R. R.—LASTOVETSKY, A.: Bi-Objective Optimization of Data-Parallel Applications on Homogeneous Multicore Clusters for Performance and Energy. *IEEE Transactions on Computers*, Vol. 67, 2018, No. 2, pp. 160–177, doi: 10.1109/TC.2017.2742513.
- [46] O'BRIEN, K.—PIETRI, I.—REDDY, R.—LASTOVETSKY, A.—SAKELLARIOU, R.: A Survey of Power and Energy Predictive Models in HPC Systems and Applications. *ACM Computing Surveys*, Vol. 50, 2017, No. 3, Art.No. 37, pp. 1–38, doi: 10.1145/3078811.
- [47] DURAN, A.—CELEBI, M. S.—PISKIN, S.—TUNCEL, M.: Scalability of OpenFOAM for Bio-Medical Flow Simulations. *The Journal of Supercomputing*, Vol. 71, 2015, No. 3, pp. 938–951, doi: 10.1007/s11227-014-1344-1.
- [48] KAČENIAUSKAS, A.—KAČIANAUSKAS, R.—MAKNICKAS, A.—MARKAUSKAS, D.: Computation and Visualization of Discrete Particle Systems on gLite-Based Grid. *Advances in Engineering Software*, Vol. 42, 2011, No. 5, pp. 237–246, doi: 10.1016/j.advengsoft.2011.02.007.
- [49] MARKAUSKAS, D.—KAČENIAUSKAS, A.: The Comparison of Two Domain Repartitioning Methods Used for Parallel Discrete Element Computations of the Hopper Discharge. *Advances in Engineering Software*, Vol. 84, 2015, pp. 68–76, doi: 10.1016/j.advengsoft.2014.12.002.
- [50] STAŠKŪNIENĖ, M.—KAČENIAUSKAS, A.—MAKNICKAS, A.—STARIKOVIČIUS, V.—STUPAK, E.—PACEVIČ, R.: Investigation of the Backflows and Outlet Boundary Conditions for Computations of the Patient-Specific Aortic Valve Flows. *Technology and Health Care*, Vol. 26, 2018, No. S2, pp. 553–563, doi: 10.3233/THC-182502.
- [51] WOLF, I.—NOLDEN, M.—BÖTTGER, T.—WEGNER, I.—SCHÖBINGER, M.—HASTENTEUFEL, M.—HEIMANN, T.—MEINZER, H.-P.—VETTER, M.: The MITK Approach. *The Insight Journal – 2005 MICCAI Open-Source Workshop*, 2005. Available at: <http://hdl.handle.net/1926/14>.
- [52] ACHESON, D. J.: *Elementary Fluid Dynamics*. Oxford University Press, 1990.
- [53] ANSYS: *ANSYS Fluent Theory Guide*. 2016.
- [54] STUPAK, E.—KAČIANAUSKAS, R.—KAČENIAUSKAS, A.—STARIKOVIČIUS, V.—MAKNICKAS, A.—PACEVIČ, R.—STAŠKŪNIENĖ, M.—DAVIDAVIČIUS, G.—AIDIETIS, A.: The Geometric Model-Based Patient-Specific Simulations of Turbulent Aortic Valve Flows. *Archives of Mechanics*, Vol. 69, 2017, No. 4-5, pp. 317–345.
- [55] CHEN, Z. J.—PRZEKAS, A. J.: A Coupled Pressure-Based Computational Method for Incompressible/Compressible Flows. *Journal of Computational Physics*, Vol. 229, 2010, No. 24, pp. 9150–9165, doi: 10.1016/j.jcp.2010.08.029.
- [56] ISSA, R. I.: Solution of the Implicitly Discretised Fluid Flow Equations by Operator-Splitting. *Journal of Computational Physics*, Vol. 62, 1986, No. 1, pp. 40–65, doi: 10.1016/0021-9991(86)90099-9.
- [57] JClouds. 2019, available at: <http://www.jclouds.org>.

- [58] JOHNSON, H.—McCORMICK, M.—IBANEZ, L.: The ITK Software Guide. Insight Software Consortium, 2014, 804 pp.
- [59] SCHROEDER, W.—MARTIN, K.—LORENSEN, B.: The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics. Kitware Inc., 2006, 528 pp.
- [60] KAČENIAUSKAS, A.—PACEVIČ, R.—BUGAJEV, A.—KATKEVIČIUS, T.: Efficient Visualization by Using ParaView Software on BalticGrid. *Information Technology and Control*, Vol. 39, 2010, No. 2, pp. 108–115.
- [61] PACEVIČ, R.—KAČENIAUSKAS, A.: The Development of VisLT Visualization Service in OpenStack Cloud Infrastructure. *Advances in Engineering Software*, Vol. 103, 2017, pp. 46–56, doi: 10.1016/j.advengsoft.2016.06.012.
- [62] STARIKOVIČIUS, V.—ČIEGIS, R.—BUGAJEV, A.: On Efficiency Analysis of the OpenFOAM-Based Parallel Solver for Simulation of Heat Transfer in and Around the Electrical Power Cables. *Informatica*, Vol. 27, 2016, No. 1, pp. 161–178, doi: 10.15388/Informatica.2016.80.
- [63] KARYPIS, G.—KUMAR, V.: Parallel Multilevel Series k -Way Partitioning Scheme for Irregular Graphs. *SIAM Review*, Vol. 41, 1999, No. 2, pp. 278–300, doi: 10.1137/S0036144598334138.



Oleg BYSTROV is Ph.D. student in informatics engineering at the Vilnius Gediminas Technical University (VGTU), Lithuania, and System Administrator at the Laboratory of Security of Information Technologies. He received his M.Sc. in informatics engineering from VGTU in 2010. His research interests include distributed and cloud computing, green computing, performance evaluation, virtualization technologies, IaaS, OpenStack, Docker, LXC, operating systems and networking.



Arnas KAČENIAUSKAS is the Director of the Institute of Applied Computer Science of Vilnius Gediminas Technical University (VGTU), Lithuania, and the Chief Researcher at the Laboratory of Parallel Computing. He also is Professor and Ph.D. supervisor at the Department of Graphical Systems at VGTU. He received his professorship in informatics engineering and Ph.D. in mechanical engineering at VGTU. He is R & D Project Manager, author and co-author of 34 scientific papers in journals indexed in Clarivate Analytics WoS database. His research interests include parallel, distributed, grid and cloud computing,

high-performance computing, performance of SaaS, Linux containers, CFD, haemodynamics, coupled problems in multiphysics, GPGPU.



Ruslan PACEVIČ is Associated Professor at the Department of Graphical Systems of Vilnius Gediminas Technical University (VGTU), Lithuania. He also is the postdoctoral research fellow at the Department of Applied Informatics of Kaunas University of Technology. He received his Ph.D. in informatics engineering from VGTU in 2015. He is the co-author of several scientific papers and participant of several European and national research projects. His research interests include distributed, grid and cloud computing, development of SaaS and middleware components, OpenStack, Eucalyptus, visualization software, GPGPU, OpenCL.



Vadimas STARIKOVIČIUS received his Ph.D. degree in mathematics from the Vilnius University, Lithuania in 2002. Currently, he is Professor at the Department of Mathematical Modelling and the Head of Laboratory of Parallel Computing at Vilnius Gediminas Technical University, Lithuania. His current research interests include parallel and distributed computing, performance evaluation, numerical methods for solution of partial differential equations. He has published over 24 refereed articles in these areas.



Algirdas MAKNICKAS is Senior Researcher at the Institute of Mechanics and the Head of the Laboratory of Numerical Simulations of Vilnius Gediminas Technical University (VGTU), Lithuania. He also is Professor and Ph.D. supervisor at the Departments of Biomechanical Engineering and Mechanical and Material Engineering at VGTU. He received his Ph.D. in mechanical engineering at VGTU. He is author and co-author of 23 articles and 16 proceeding papers in journals indexed in Clarivate Analytics WoS database. His research interests include biomechanical engineering, haemodynamics, geometric model-

ling, applications of high-performance computing, linear and non-linear continuum mechanics, coupled problems in multiphysics, artificial intelligence and computational complexity, GPGPU.



Eugenius STUPAK is Associated Professor at the Department of Applied Mechanics of Vilnius Gediminas Technical University (VGTU), Lithuania. He received his Ph.D. in Mechanical Engineering from VGTU in 2004. He is the co-author of several scientific papers and participant of several national research projects. His research interests include advanced mesh generation strategies, patient-specific modelling, solution of coupled multiphysics problems.



Aleksandr IGUMENOV is Lecturer in the Department of Information Technologies at Vilnius Gediminas Technical University, Lithuania. He received his Ph.D. in informatics engineering from the Vilnius University in 2012. He is the author and co-author of several scientific papers. His main research interests include green and energy efficient computing, high-performance computing, IoT and internet technologies, blockchain technologies, global optimization.

PROCESS DATA INFRASTRUCTURE AND DATA SERVICES

Reginald CUSHING, Onno VALKERING

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

Adam BELLOUM

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

Souley MADOUGOU

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

Martin BOBAK, Ondrej HABALA, Viet TRAN

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

Jan MEIZNER, Piotr NOWAKOWSKI

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

Mara GRAZIANI, Henning MÜLLER

*University of Applied Sciences of Western Switzerland
HES-SO Valais, 3960 Sierre, Switzerland*

✉

*Department of Computer Science, University of Geneva
1227 Carouge, Switzerland*

e-mail: {mara.graziani, henning.mueller}@hevs.ch

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

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

1 INTRODUCTION

We see application of HPC in both the scientific domain and industry: ranging from modeling global climate phenomenon to designing more efficient drugs. The state of the art in HPC is at the petascale (in the order of 10¹⁵ FLOPS), first achieved in 2008 [33]. However, we now see an enormous increase in the size of commercial and scientific datasets. Consequently, it is likely that the current petascale technologies

will not be able to handle this and that we will require new solutions to prepare ourselves for the next milestone: exascale computing (1018 FLOPS). Exascale systems are expected to be realized by 2023 and will likely comprise of 100 000 interconnected servers; simply scaling up petascale solutions will likely not suffice [35]. Evidently, this raises many challenges in how the required hardware but also how to design applications that can make use of that many computing nodes. However, what is relevant for this paper is how the large volumes of data involved will be stored. Exascale systems will need novel Distributed File Systems (DFS), sufficiently scalable to accommodate for established DFS solutions.

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

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

In this paper, we describe in detail the PROCESS Data Infrastructure and Data Services which aim to be a milestone in the path in the era of exascale computing. To put this work in context, in Section 2 we provide an overview of several established state-of-the-art solutions while also highlighting novel research projects and

regarding them in an exascale computing context. In Section 3, we describe the architecture design processing starting from the requirement analysis, architectural decision and technology choices. In Section 4, we describe the implementation of the main important data services: LOBCDER, DataNet, and DISPEL, as well as all the mechanisms and APIs needed for their interactions. Finally, in Section 5 we describe the applicability to the different PROCESS use cases and derive a couple of scenarios.

2 RELATED WORK

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

2.1 Scalability of Metadata Management

To ensure a future use of current DFS designs means that they not only have to be scalable in terms of actual storage capacity, but also in metadata management. The metadata scalability is of an extreme importance as half of the data processing operations are metadata operations [40], however, we have observed a lack of metadata scalability in most of the well-known DFS. Design of GFS and HDFS feature a single metadata server, Lustre allows for multiple metadata servers, but relies on explicitly storing the locations of files. GlusterFS somewhat improves in this aspect by not explicitly storing metadata regarding file locations but opting for algorithmic placement instead. It must be noted however that even with this in place, all the other metadata operations still happen on the data storage servers. The design of Ceph is probably the most scalable with respect to metadata management, since it allows for a cluster of metadata servers and also features algorithmic file placement and dynamic metadata workload distribution. A notable recent development in relational databases is NewSQL, a class of databases seeking to combine the scalability characteristics of NoSQL databases with the transactional characteristics of traditional relational databases. In a 2017 paper Niazi et al. present HopFS, a DFS built on top of HDFS, replacing the single metadata server with a cluster of NewSQL databases storing the metadata [41]. They attempt to address the issue of metadata management scalability by storing all HDFS metadata in a Network Database (NDB), a NewSQL engine for MySQL Cluster. They tested their solution on a Spotify workload (a Hadoop cluster of 1600+ servers storing 60 petabytes of data) for which they observed a throughput increase of 16–37× compared to regular HDFS. What makes this solution noteworthy is that it is a drop-in replacement for HDFS, allowing to be used in existing Hadoop environments, allowing them to scale beyond the limits imposed by the single metadata server approach. Using a similar

approach, Takatsu et al. present PPFS (Post-Petascale File System), a DFS optimized for high file creation workloads. In their paper they argue that modern DFSs are not optimized for high file creation workloads, and that for exascale computing this can turn out to be a serious performance bottleneck [42]. They have evaluated their system against IndexFS (2014), a middleware for file systems such as HDFS and Lustre aiming to improve metadata performance [43]. With respect to file creation performance, they observed a $2.6\times$ increase in performance. They achieved this by employing a distributed metadata cluster design using key-value metadata storage and non-blocking distributed transactions to simultaneously update multiple entries. Although only tested on relatively small clusters comprising of tens of servers, it is good to see that an effort is being made to improve upon aspects such as file creation performance, which might be a bottleneck in an exascale context.

2.2 Decentralization

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

What are advantages to the design of GlusterFS? First of all, it offers POSIX file semantics, which means that it is mountable like any other traditional file system and adheres to strict consistency requirements. Secondly, its replication via erasure codes is a more space efficient way of replicating data than naively storing multiple copies. But the main advantage is the fact that the design does not feature a server explicitly storing file location metadata. With respect to scalability, not requiring a metadata server that can potentially be a performance bottleneck is a significant

benefit. For certain workloads, a disadvantage of the design of GlusterFS is that it works on file granularity (as opposed to aggregated data blocks or chunks). Such a design can introduce more internal administrative overhead when for example replicating huge numbers of small files. However, we deem it likely that its approach of having a decentralized Namespace will manifest itself in exascale DFS solutions of the future. The design of Ceph, allowing for clusters of not only data, but also metadata and monitor servers provides it with excellent scalability characteristics. Currently, it is already being used by Yahoo to store petabytes of data and is chosen as the technology to prepare their infrastructure for storing exabytes of data [45]. This in combination with the level of customizability makes Ceph a good candidate for an exascale computing DFS.

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

3 DESIGN OF THE PROCESS DATA INFRASTRUCTURE

3.1 Requirements

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

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

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

Table 1. Main data characteristics of the use cases

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

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

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

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

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

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

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

3.2 Design

3.2.1 Process Data System

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

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

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

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

Figure 1 illustrates the notion of a micro-infrastructure. Site providers provide raw resources through virtualization middleware such as OpenStack. They also provide raw storage that is accessible through the virtual machines. Through templating, micro-infrastructures can be booted up that will satisfy the groups’ requirements for data processing. Cross provider data, process distribution and management are handled from within the micro-infrastructure. Cross group collaboration is also

easily manageable, e.g., a group could give access to another group through their ownCloud node inside the micro-infrastructure. Scalability is improved since state management is divided between micro-infrastructures. One data management system will have difficulty managing exascale data, but many micro-infrastructures can better manage their own pool of data which is, most often, a few orders of magnitude less than an exabyte.

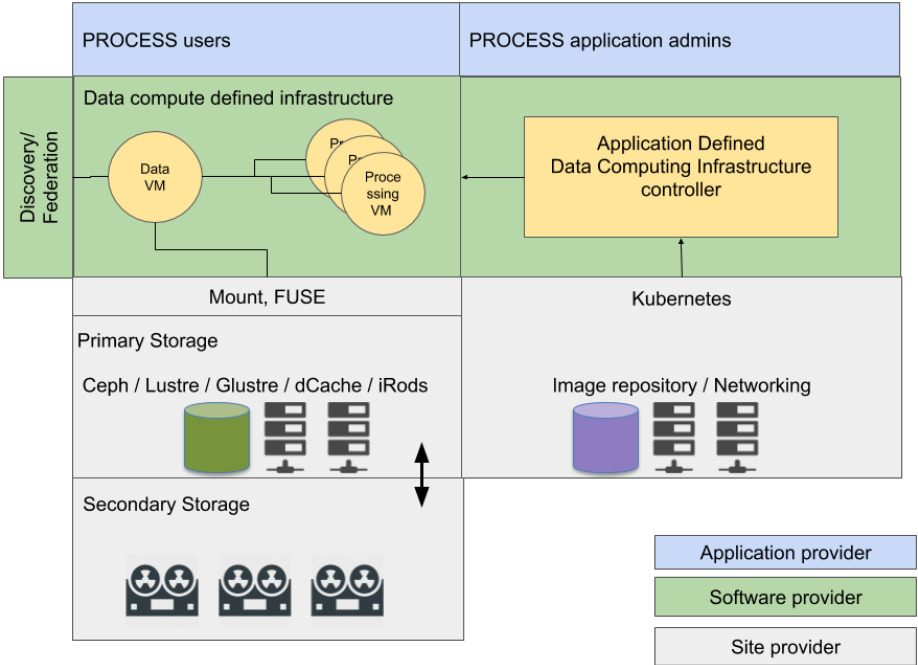


Figure 1. PROCESS micro-infrastructure

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

4 IMPLEMENTATION

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

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

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

4.1 LOBCDER/Micro-Infrastructure

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

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

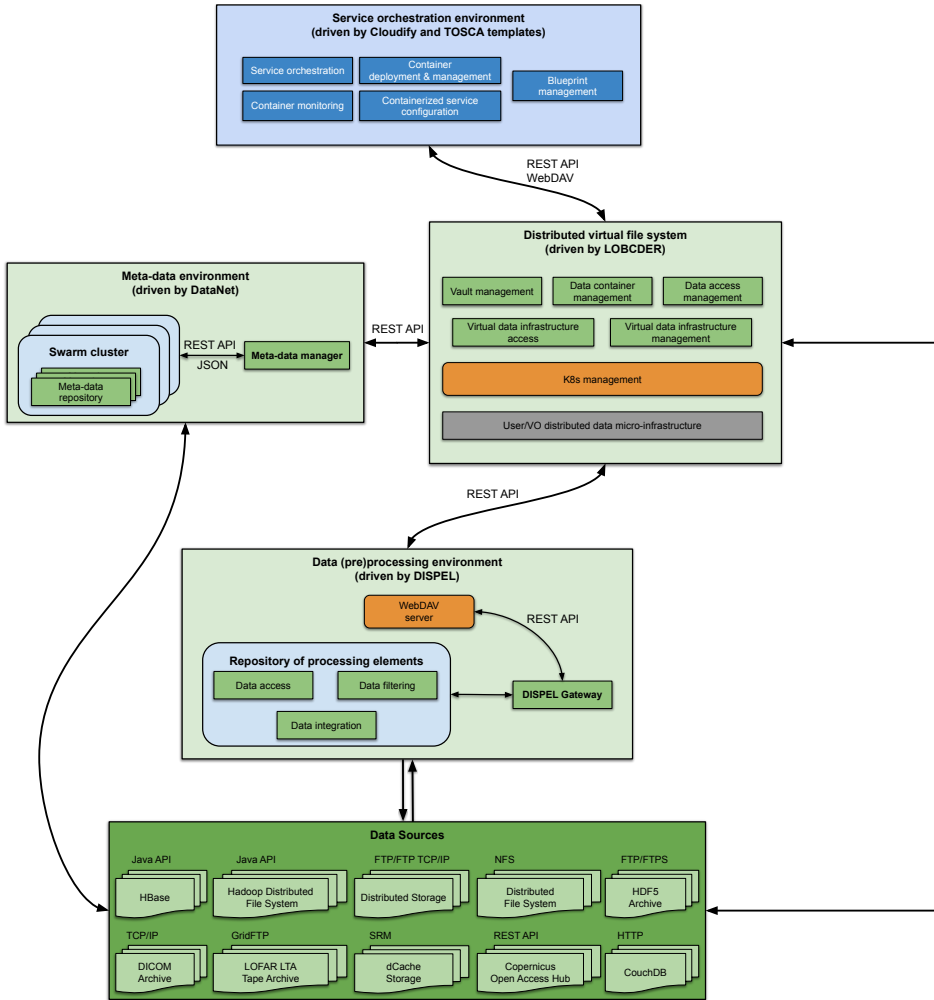


Figure 2. Process data service environment

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

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

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

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

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

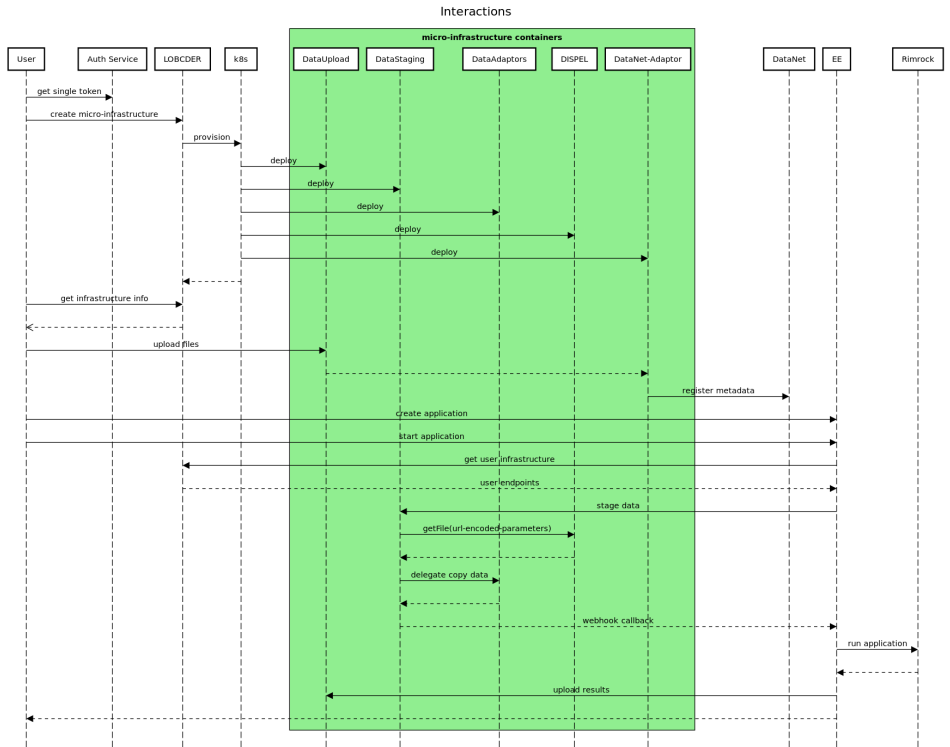


Figure 3. Sequence of interacting components from user perspective. The green block is a dynamically created virtual infrastructure per use-case. The infrastructure encapsulates use-cases' data management, credentials, distributed resources and pre-processing routines.

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

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

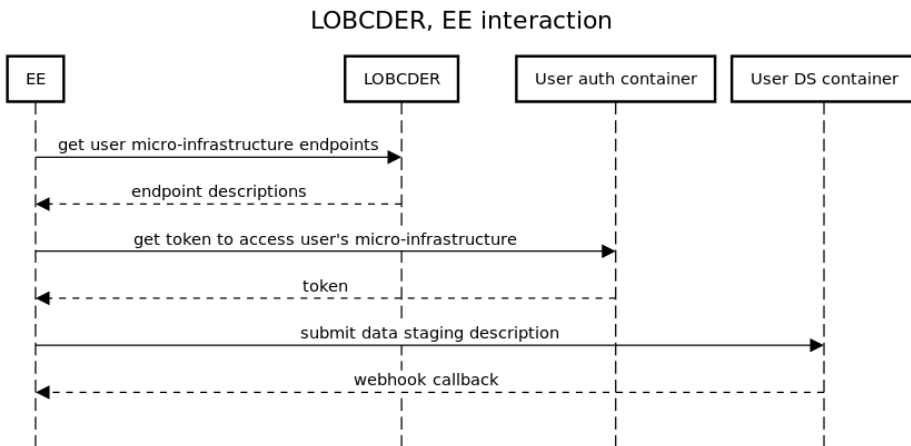


Figure 4. Simplified EE to LOBCDER interaction. EE queries LOBCDER to retrieve user's dynamic infrastructure. EE can then access user's data services, e.g. data staging.

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

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

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

4.2 Storage Adapter Containers

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

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

Table 4. Three storage adapter containers

4.3 Logic Containers

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

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

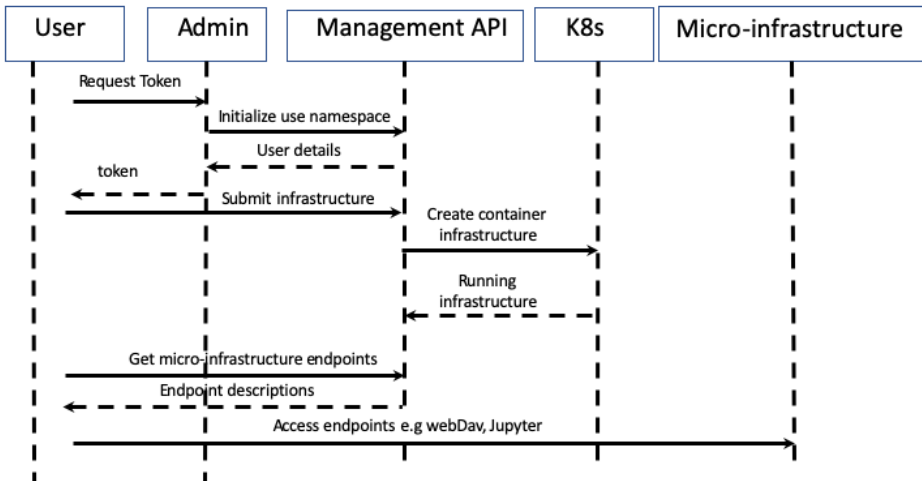


Figure 5. Simplified user to LOBCDER interaction. User requests credentials to access LOBCDER API. User can then submit requests to create virtual infrastructure and access services.

5 APPLICATION USE CASES

5.1 Process Data Service in Medical Use Case

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

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

⋮

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

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

⋮

DISPEL	<p>DISPEL container gives access to remotely stored data and an entire data (pre)processing environment. The DISPEL data processing environment is currently available as a Debian-based virtual machine with a complete deployment of all tools, manuals and a tutorial with example data processes. The VM contains a graphical development environment based on Eclipse. Dispel is accessed via standard WebDAV interface (HTTP protocol) since it is part of the LOBCDER distributed data infrastructure. The parameters for data processing are encoded in the provided HTTP URL, as described previously in D5.1. The HTTP URL encodes the following parameters: (1) One selection of DISPEL data process description file template (in the DISPEL language). (2) Zero or more parameters to be filled in the template. Example of URL-encoded parameters: <code>http://lobcder.process-project.eu/dispel/tiffstore/312/20181129/12/0-1200-0-400</code>. Components of the URL are:</p> <ul style="list-style-type: none">• <code>http://lobcder.process-project.eu/</code>: URL of the LOBCDER WebDAV server• <code>dispel</code>: a prefix to recognize the sub-repository to contact (the DISPEL service)• <code>tiffstore</code>: selection of the DISPEL data process template to execute• <code>312</code>: subject designation (application-specific metadata)• <code>20181129</code>: data creation time (application-specific metadata)• <code>12</code>: layer in a multi-layer TIFF file (application-specific metadata)• <code>0-1200-0-400</code>: grid selection (application-specific metadata)
DataNet-adaptor	<p>DataNet-adaptor container allows pushing of metadata to the Datanet service. DataNet allows performing operations on the metadata sets such as creating/updating/querying/deleting entities. DataNet is designed to offer straightforward user access via the REST API as well as GUI HAL browser.</p> <p>DataNet is available in the form of the Java source code under the OSI approved license as well as a Docker Container for the convenient deployment DataNet Rest API is described in Annex C</p>
NextCloud [27]	<p>Service for ease of use by users. Next Cloud container allows the user to view their data in dropbox fashion.</p>

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

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

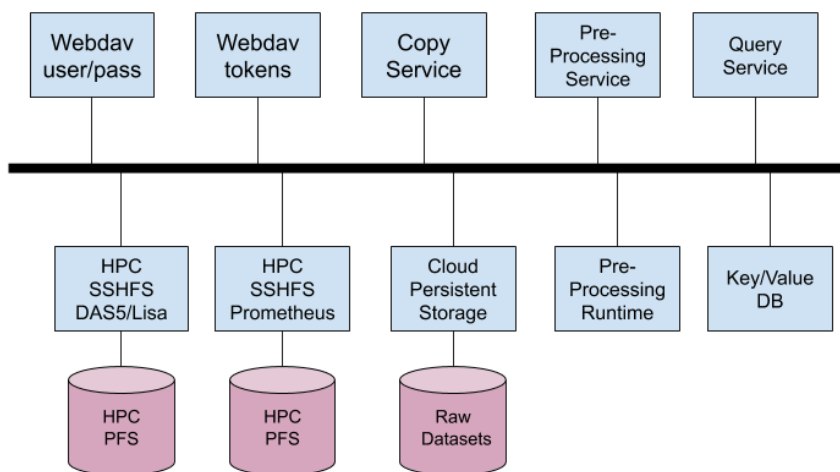


Figure 6. Micro-infrastructure for learning on medical image use case

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

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

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

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

Pre-processing service: This REST service container will allow users to define input raw data, input hyper-parameters to generate new pre-processed datasets and HPC output locations so that the pre-processed datasets are pushed directly

onto the HPC file systems. It also keeps track of these generated datasets using a local database.

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

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

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

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

5.2 Data Services for LOFAR Use Case

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

PROCESS can offer access to the three sites where the LOFAR Long Term Archive is stored – Amsterdam, Jülich and Poznan, making simultaneous combined staging and processing of observations possible. Presently, processing of data from

different sites requires separate user interfaces. The services required for the LOFAR Use Case pipeline are shown in Figure 7.

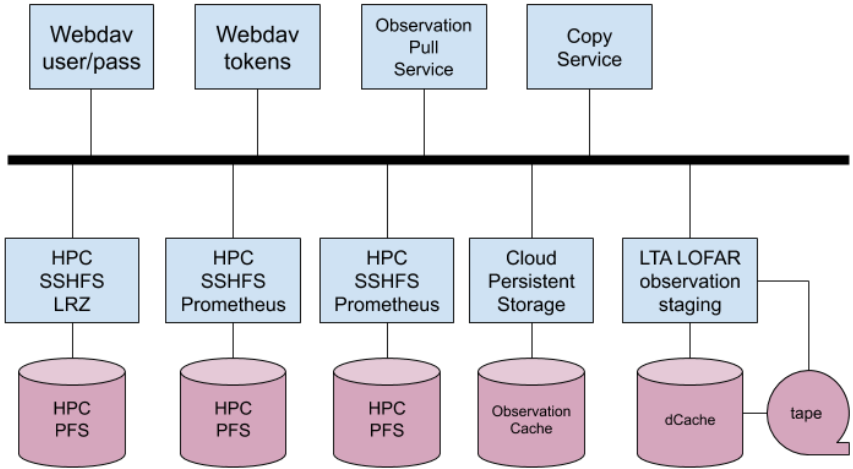


Figure 7. UC#2: Data infrastructure including data adapters as well as long-term-archive staging service

5.3 Experiments

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

5.3.1 Process Data Service in Medical Use Case

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

Table 6 shows the cross site staging part of the Camelyon16 dataset using several protocols and strategies. Our initial approach is to use the widely available SCP protocol and use cluster head nodes to stage the data. From the table, this approach is shown to be one of the worst strategies, e.g. LISA to AGH. Furthermore, head nodes are very unstable for staging such data with frequent stalls and broken connections. A second strategy is to use Data Transfer Nodes (DTN). What we can show from the tests is that using such DTN nodes can speedup transfers,

for example, transferring data from LRZ site to LISA is faster by using a DTN relay than by a direct copy. E.g. LRZ-VDTN to AMS-DTN takes 5.85 minutes plus AMS-DTN to LISA is 3.03 minutes, cumulatively it is 8.9 minutes which is $\sim 30\%$ faster than a direct copy which is 12.96 minutes. The results also show the added speedup by using campus networks. The AMS-DTN and LISA are on campus thus their bandwidth is approximately 4 times better than the rest. Using DTNs as cache staging nodes could potentially accelerate data transfers between sites. With these tests we feel that we have a basis for the upcoming steps for UC1. Moreover, the additional requirement of SCP protocol for data transfer as stated in D4.3 page 9 has been met. This ensures that data transfer can be applied to different types of users, and especially hospital institutions which may not have open FTP access for security reasons.

Protocol – SCP	30 GB Came- lyon16.partAA [MB/s]	30 GB Came- lyon16.partAB [MB/s]	30 GB Came- lyon16.partAC [MB/s]	Mean BW [MB/s]	Duration [min- utes]
LRZ-V- DTN to AMS- DTN	86.8	90.8	84.9	87.5	5.85
LRZ-V- DTN to AGH	33.1	31.2	32.2	32.17	15.92
LRZ-V- DTN to LISA	25.5	64	29	39.5	12.96
AMS- DTN to LISA	167.9	172.6	166	168.83	3.03
AMS- DTN to AGH	53	53.9	38.9	48.6	10.53
LISA to AGH	40	21.3	29.5	30.27	16.91

Table 6. Data transfer tests between different storage locations

5.3.2 Data Services in the LOFAR Use Case

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

speeds from the three LTA locations to each of the three PROCESS HPC clusters; and execution time as a function of total size.

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

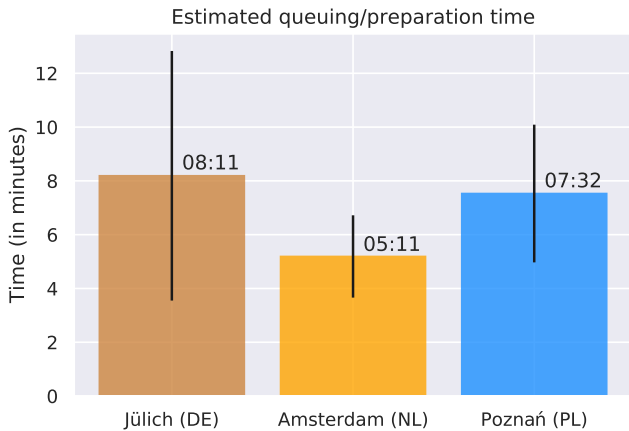


Figure 8. Estimated queuing/preparation time

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

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

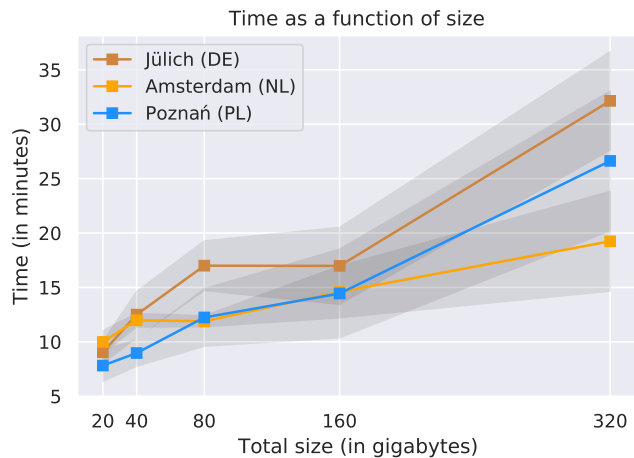


Figure 9. Staging time as a function of size

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

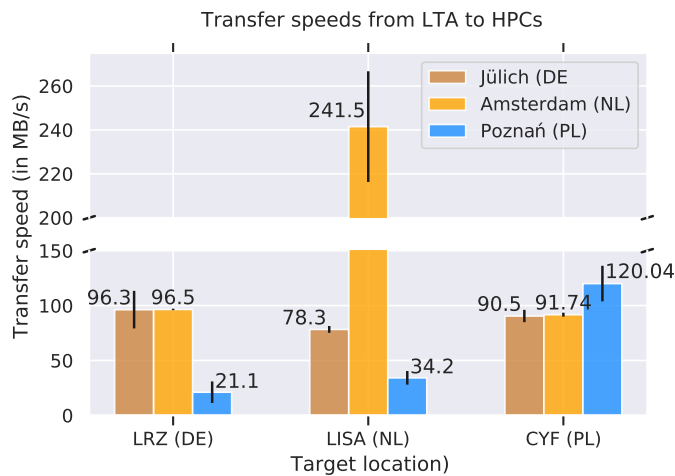


Figure 10. Transfer speeds from LTA to HPCs

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

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

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

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

6 CONCLUSIONS AND FUTURE WORK

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

their specific ones. Because of the container centric approach followed in the current implementation of the micro-infrastructure, it was straightforward to mix and match various data adapters to fulfill the applications data handling requirements. All software components developed and reported in this paper are available in the PROCESS software repository [56].

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

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

Acknowledgment

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

REFERENCES

- [1] KOEHLER, M.—KNIGHT, R.—BENKNER, S.—KANIOVSKYI, Y.—WOOD, S.: The VPH-Share Data Management Platform: Enabling Collaborative Data Management for the Virtual Physiological Human Community. 2012 Eighth International Conference on Semantics, Knowledge and Grids, Beijing, China, 2012, pp. 80–87, doi: 10.1109/SKG.2012.51.
- [2] BENT, J.—FAIBISH, S.—AHRENS, J.—GRIDER, G.—PATCHETT, J.—TZELNIC, P.—WOODRING, J.: Jitter-Free Co-Processing on a Prototype Exascale

- Storage Stack. 2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), San Diego, CA, USA, 2012, pp. 1–5, doi: 10.1109/MSST.2012.6232382.
- [3] FILIPPIDIS, C.: Parallel Storage Systems for Large Scale Machines. http://sc16.supercomputing.org/sc-archive/doctoral_showcase/doc_files/drs104s2-file2.pdf.
 - [4] LOFSTEAD, J.—JIMENEZ, I.—MALTZAHN, C.—KOZIOL, Q.—BENT, J.—BARTON, E.: DAOS and Friends: A Proposal for an Exascale Storage System. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'16), Salt Lake City, UT, USA, IEEE, 2016, pp. 585–596, doi: 10.1109/SC.2016.49.
 - [5] HEMSOTH, N.: Exascale Storage Gets a GPU Boost DFAF. 2018, <https://www.nextplatform.com/2018/02/12/exascale-storage-gets-gpu-boost/>.
 - [6] HEMSOTH, N.: An Exascale Timeline for Storage and I/O System. 2017, <https://www.nextplatform.com/2017/08/16/exascale-timeline-storage-io-systems/>.
 - [7] Infinite Memory Engine: The Exascale-Era Storage Architecture. 2017, <https://www.hpcwire.com/2017/08/21/infinite-memory-engine-exascale-era-storage-architecture/>.
 - [8] HICK, J.—WATSON, D.—COOK, D.—MINTON, J.—NEWMAN, H.—PRESTON, T.—RICH, G.—SCOTT, C.—SHOOPMAN, J.—NOE, J.—O'CONNELL, J.—SHIPMAN, G.—WHITE, V.: HPSS in the Extreme Scale Era: Report to DOE Office of Science on HPSS in 2018–2022. Technical Report No. LBNL-3877E, Lawrence Berkeley National Lab. (LBNL), Berkeley, CA, USA, 2009.
 - [9] COPE, J.—LIU, N.—LANG, S.—CARNS, P.—CAROTHERS, C.—ROSS, R.: CODES: Enabling Co-Design of Multi-Layer Exascale Storage Architectures. <https://pdfs.semanticscholar.org/159d/bd0a8c18e2df895b131e33499e2d529210e0.pdf>.
 - [10] MAIR, J.—HUANG, Z.—EYERS, D.—CHEN, Y.: Quantifying the Energy Efficiency Challenges of Achieving Exascale Computing. 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, China, 2015, pp. 943–950, doi: 10.1109/CCGrid.2015.130.
 - [11] CAMERON, K.W.: Energy Efficiency in the Wild: Why Datacenters Fear Power Management. *Computer*, Vol. 47, 2014, No. 11, pp. 89–92, doi: 10.1109/MC.2014.315.
 - [12] <https://www.eudat.eu/services/b2stage>.
 - [13] <https://www.dcache.org>.
 - [14] <https://www.gluster.org>.
 - [15] <https://iRODS.org>.
 - [16] <https://www.dcache.org/manuals/2016/presentations/20161006-PM-dCache.pdf>.
 - [17] SHANNIGRAHI, S.—FAN, C.—PAPADOPOULOS, C.: Named Data Networking Strategies for Improving Large Scientific Data Transfers. Proceedings of the IEEE International Conference on Communications Workshops (ICC Workshops), Kansas City, MO, USA, 2018, doi: 10.1109/ICCW.2018.8403576.

- [18] CHEN, S.—CAO, J.—ZHU, L.: NDSS: A Named Data Storage System. 2015 International Conference on Cloud and Autonomic Computing, Boston, MA, USA, 2015, pp. 196–199, doi: 10.1109/ICCAC.2015.12.
- [19] ZHU, S.—YUAN, M.—LEI, K.: Ndynamic: An ndnDHT-Based Distributed Storage System over Named Data Networking. 2016 5th International Conference on Computer Science and Network Technology (ICCSNT), Changchun, China, 2016, pp. 148–152, doi: 10.1109/ICCSNT.2016.8070137.
- [20] RAO, Y.—GAO, D.—ZHANG, H.—FOH, C. H.: Mobility Support for the User in NDN-Based Cloud Storage Service. 2015 IEEE Globecom Workshops (GC Wkshps), San Diego, CA, USA, 2015, pp. 1–6, doi: 10.1109/GLOCOMW.2015.7414159.
- [21] Cloudify Orchestration Service. <https://docs.cloudify.co/latest/developer/apis/>.
- [22] ATKINSON, M. P.—GALEA, M.—LIEW, C. S.—MARTIN, P.: ADMIRE – Final Report on the ADMIRE Architecture, with an Assessment and Proposals for Its Development. Technical Report, The ADMIRE Project, May 2011.
- [23] BREZANY, P.—ARANDA, C. B.—CORCHO, O.—JANCIK, I.—WOEHRER, A.—ATKINSON M.: ADMIRE – Report Defining the Final Iteration of the Model and Language. Deliverable Report D1.9, The ADMIRE Project, May 2011.
- [24] Available at GitHub: <https://github.com/micro-infrastructure/adaptor-sshfs>.
- [25] Available at GitHub: <https://github.com/micro-infrastructure/service-webdavserver>.
- [26] Available at GitHub: <https://github.com/micro-infrastructure/service-scp2scp>.
- [27] Available at GitHub: <https://github.com/micro-infrastructure/service-nextcloud>.
- [28] GRAZIANI, M.—EGGEL, I.—DELIGAND, F.—BOBÁK, M.—ANDREARCZYK, V.—MÜLLER, H.: Breast Histopathology with High-Performance Computing and Deep Learning. *Computing and Informatics*, Vol. 39, 2020, No. 4, pp. 780–807, doi: 10.31577/cai.2020.4.780.
- [29] <https://www.astron.nl/lofarwiki/doku.php?id=start>.
- [30] <https://www.surf.nl/en/use-case-space-research-with-grid-infrastructure>.
- [31] <https://onnovalkering.github.io/brane/>.
- [32] <https://github.com/recap/MicroInfrastructure>.
- [33] TOP500.org. The TOP500 List (June 2008). 2008, <https://www.top500.org/lists/2008/06/>. Accessed: 04-01-2018.
- [34] BÁNDI, P.—GEESING, O.—MANSON, Q.—VAN DIJK, M.—BALKENHOL, M.: From Detection of Individual Metastases to Classification of Lymph Node Status at the Patient Level: The CAMELYON17 Challenge. *IEEE Transactions on Medical Imaging*, Vol. 38, 2019, No. 2, pp. 550–560, doi: 10.1109/TMI.2018.2867350.
- [35] VIJAYARAGHAVAN, T.—ECKERT, Y.—LOH, G. H.—SCHULTE, M. J.—IGNATOWSKI, M.—BECKMANN, B. M.—BRANTLEY, W. C.—GREATHOUSE, J. L.—HUANG, W.—KARUNANITHI, A. et al.: Design and Analysis

- of an APU for Exascale Computing. 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), IEEE, 2017, pp. 85–96, doi: 10.1109/HPCA.2017.42.
- [36] HGST Ultrastar Hs14. <http://www.hgst.com/products/harddrives/ultrastar-hs14>. Accessed: 03-12-2017.
- [37] ASTC Technology Roadmap. <http://idema.org/?pageid=5868>. Accessed: 03-12-2017.
- [38] SANDBERG, R.—GOLDBERG, D.—KLEIMAN, S.—WALSH, D.—LYON, B.: Design and Implementation of the Sun Network File System. Proceedings of the Summer 1985 USENIX Conference, Portland, OR, USA, 1985, pp. 119–130.
- [39] LEVY, E.—SILBERSCHATZ, A.: Distributed File Systems: Concepts and Examples. ACM Computing Surveys (CSUR), Vol. 22, 1990, No. 4, pp. 321–374, doi: 10.1145/98163.98169.
- [40] ROSELLI, D. S.—LORCH, J. R.—ANDERSON, T. E.: A Comparison of File System Workloads. Proceedings of the USENIX Annual Technical Conference, General Track, 2000, pp. 41–54.
- [41] NIAZI, S.—ISMAIL, M.—HARIDI, S.—DOWLING, J.—GROHSSCHMIEDT, S.—RONSTRÖM, M.: HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases. Proceedings of the 15th Usenix Conference on File and Storage Technologies (FAST 2017), Santa Clara, CA, USA, pp. 89–103.
- [42] TAKATSU, F.—HIRAGA, K.—TATEBE, O.: PPFs: A Scale-Out Distributed File System for Post-Petascale Systems. Journal of Information Processing, Vol. 25, 2017, pp. 438–447, doi: 10.2197/ipsjip.25.438.
- [43] REN, K.—ZHENG, Q.—PATIL, S.—GIBSON, G.: IndexFS: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC’14), IEEE, 2014, pp. 237–248, doi: 10.1109/SC.2014.25.
- [44] BENET, J.: IPFS – Content Addressed, Versioned, P2P File System. arXiv preprint arXiv:1407.3561, 2014.
- [45] Yahoo Cloud Object Store – Object Storage at Exabyte Scale. <https://yahooeng.tumblr.com/post/116391291701/yahooncloud-object-store-object-storage-at>. Accessed: 06-01-2018.
- [46] KOULOUZIS, S.—BELLOUM, A. S. Z.—BUBAK, M. T.—ZHAO, Z.—ŽIVKOVIĆ, M.—DE LAAT, C. T. A. M.: SDN-Aware Federation of Distributed Data. Future Generation Computer Systems, Vol. 56, 2016, pp. 64–76, doi: 10.1016/j.future.2015.09.032.
- [47] KOULOUZIS, S.—BELLOUM, A.—BUBAK, M.—LAMATA, P.—NOLTE, D.—VASYUNIN, D.—DE LAAT, C.: Distributed Data Management Service for VPH Applications. IEEE Internet Computing, Vol. 20, 2016, No. 2, pp. 34–41, doi: 10.1109/MIC.2015.71.
- [48] <https://www.nist.gov/publications/nist-big-data-interoperability-framework-volume-1-definitions>.
- [49] <https://lta.lofar.eu>.

- [50] https://www.process-project.eu/wp-content/uploads/2020/02/PUBLIC_PROCESS_D4.1_Initial_-state_of_the_art_and_requirement_analysis_initial_PROCESS_architecture_v1-1.pdf.
- [51] https://www.process-project.eu/wp-content/uploads/2020/02/PUBLIC_PROCESS_D4.2_Report_on_architecture_v1.0.pdf.
- [52] https://www.process-project.eu/wp-content/uploads/2020/02/PUBLIC_PROCESS_D5.1_Design_of_data_infrastructure_v1.0.pdf.
- [53] https://www.process-project.eu/wp-content/uploads/2020/02/PROCESS_D5.2_Alpha_release_of_the_Data_service_v1.0.pdf.
- [54] SPREEUW, H.—MADOUGOU, S.—VAN HAREN, R.—WEEL, B.—BELLOUM, A.—MAASSEN, J.: Unlocking the LOFAR LTA. 2019 15th International Conference on eScience (eScience), San Diego, CA, USA, 2019, pp. 467–470, doi: 10.1109/eScience.2019.00061.
- [55] https://www.process-project.eu/wp-content/uploads/2020/02/PUBLIC_PROCESS_D2.1_Progress_Report_v1.0.pdf.
- [56] <https://www.research-software.nl/>.
- [57] WITTENBURG, P.—VAN DE SOMPEL, H.—VIGEN, J.—BACHEM, A.—ROMARY, L.—MARINUCCI, M.—ANDERSSON, T.—GENOVA, F.—BEST, C.—LOS, W. et al.: Riding the Wave: How Europe Can Gain from the Rising Tide of Scientific Data. Final Report of the High Level Expert Group on Scientific Data – A Submission to the European Commission, October 2010. http://ec.europa.eu/newsroom/dae/document.cfm?doc_id=707.



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



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



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



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



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



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



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

author or co-author of over 100 scientific publications.



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



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



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

and language at the University of Cambridge, UK in 2017.



Henning MÜLLER is Full Professor at the HES-SO Valais and responsible for the eHealth unit of the school. He is also Professor at the Medical Faculty of the University of Geneva and has been on sabbatical at the Martinos Center, part of Harvard Medical School in Boston, MA, USA to focus on research activities. He is the coordinator of the ExaMode EU project, was the coordinator of the Khresmoi EU project, the scientific coordinator of the VISCERAL EU project, and is the initiator of the ImageCLEF benchmark that has run medical tasks since 2004.

He has authored over 500 scientific papers with more than 13 000 citations and is in the editorial board of several journals.

DISTRIBUTED ALGORITHM FOR PARALLEL EDIT DISTANCE COMPUTATION

Muhammad Umair SADIQ, Muhammad Murtaza YOUSAF

Punjab University College of Information Technology (PUCIT)

University of the Punjab

Lahore, Pakistan

e-mail: {umair.sadiq, murtaza}@pucit.edu.pk

Abstract. The edit distance is the measure that quantifies the difference between two strings. It is an important concept because it has its usage in many domains such as natural language processing, spell checking, genome matching, and pattern recognition. Edit distance is also known as Levenshtein distance. Sequentially, the edit distance is computed by using dynamic programming based strategy that may not provide results in reasonable time when input strings are large. In this work, a distributed algorithm is presented for parallel edit distance computation. The proposed algorithm is both time and space efficient. It is evaluated on a hybrid setup of distributed and shared memory systems. Results suggest that the proposed algorithm achieves significant performance gain over the existing parallel approach.

Keywords: Edit distance, dynamic programming, parallel computing, distributed memory system, MPI, OpenMP, speedup

1 INTRODUCTION

Measuring the similarity between two strings helps to solve problems in many domains such as spell checking, spam filtering, nucleotide sequence matching, virus signature matching in computer security, natural language processing (NLP), speech recognition, and pattern recognition [1, 2, 3, 4, 5]. String similarity/matching comes in two forms: approximate string matching and exact string matching [6]. In the exact string matching, all the appearances of the pattern are required to be found in the given string. In the approximate string matching, the difference between the given pattern and the string is measured. Levenshtein distance is the measure that

tells the difference between two strings. It counts the number of edit operations (insert, replace, and delete) that are required to transform one string to another [7]. In literature, it is often referred to as edit distance [8, 9], but some other definitions of edit distance exist as well [10]. In this study, we would consider the Levenshtein's definition of the edit distance.

Sequentially, edit distance/Levenshtein distance can be computed by using dynamic programming based strategy but if large strings such as deoxyribonucleic acid (DNA) sequences are compared, then it may not give result in a reasonable amount of time. Therefore, a parallel solution is required to compute result in acceptable time if the data size is large. This work is about design and evaluation of a distributed algorithm for parallel edit distance computation between two strings. The proposed algorithm is evaluated on a cluster by using Message Passing Interface (MPI). MPI is designed to work with different parallel architectures and it serves as a standard [11]. It defines certain point to point and the collective communication protocols to program distributed parallel systems. The rest of the paper is organized as follows: Section 2 discusses some preliminary concepts. Section 3 presents the background of the edit distance. Section 4 covers the related work. Section 5 presents a distributed algorithm for parallel edit distance computation. Section 6 presents experimental evaluation of the proposed algorithm. Finally, Section 7 concludes the paper and discusses the artery of future work.

2 PRELIMINARIES

This section introduces some basic terminologies and key concepts that will be used in the rest of the paper.

2.1 Cost of Communication

While designing a distributed algorithm, computation, as well as communication time between different nodes, is also essential. For analysis of communication time, consider this model of communication. The time required to communicate a message between two nodes of a distributed memory system is equal to $t_s + t_w\eta$ where t_s is startup time to prepare the message for transmission, t_w is per word transfer time, and η is number of words [12].

2.2 Exclusive Scan Operation

Exclusive scan operation is an important primitive in parallel computing. It operates on an ordered set $[x_0, x_1, \dots, x_{n-1}]$ of n elements. It uses a binary associative operator \oplus . It returns the result of form: $[-, x_0 \oplus x_1, x_0 \oplus x_1 \oplus x_2, \dots, x_0 \oplus x_1 \oplus x_2 \oplus \dots \oplus x_{n-1}]$. In the output each j^{th} element is cumulative result of all input elements from 0^{th} to j^{th} element (excluding j^{th} element itself). If all n elements are divided among p processors then this operation requires $(t_s + t_w\eta) \log n$ time on distributed memory system [12].

2.3 Speedup

In parallel computing, speedup is the measure of the increase of performance of parallel algorithm compared to sequential algorithm. It is the ratio of sequential execution time to parallel execution time.

2.4 Parallel Efficiency

Parallel efficiency is the measure of effectiveness of the resource utilization. It is the ratio of speedup to the number of compute nodes.

3 SEQUENTIAL COMPUTATION OF THE EDIT DISTANCE

Sequentially, edit distance is computed by using dynamic programming based strategy in which a table *Lev* of size $(m + 1) \times (n + 1)$ is built where m is size of first string and n is size of second string [7]. Given two strings A and B of size m and n , respectively, edit distance table can be computed with Algorithm 1.

Each cell (i, j) in the edit distance table represents the value of the edit distance between the first i characters of string A and the first j characters of string B . Cell (m, n) in the table represents the value of the edit distance between both strings. Figure 1 shows a sample edit distance table for strings “ACER” and “CARE”.

Edit distance between “ACER” and “CARE” is 3 because to make this conversion following operations are required: delete ‘A’, match ‘C’, replace ‘E’ with ‘A’, match ‘R’, and delete ‘E’. There is no cost for matching a character. Figure 1 also illustrates the dependence of the calculation of a single cell on other cells of the edit distance table.

Sequential time complexity of the algorithm is $O(mn)$ which is obvious from the size of the edit distance table. Space complexity of the algorithm is $O(n)$ [7].

4 RELATED WORK

To reduce the computation time of the edit distance many efforts have been made and this section covers various such studies.

Masek and Paterson [8] presented a little restricted but fast sequential edit distance algorithm for abstract unit-cost RAM machine. It requires the strings to be of equal sizes. Mathies [9] presented a fast parallel algorithm for the edit distance computation. It requires mn processors on abstract parallel random-access machine. Apostlico et al. [13] presented a parallel edit distance algorithm for the abstract parallel random-access machine. A space efficient algorithm for the edit distance is presented in [14].

A bit parallel algorithm for the problem of approximate string matching [15] is presented in [16]. This algorithm is serial and depends upon the word size of machine. It allows to process only w cells at a time where w is the largest word

Algorithm 1: Sequential computation of the edit distance [7].

Input: Strings: $A[0 \dots m - 1]$ and $B[0 \dots n - 1]$

```

1  $m \leftarrow A.length$ 
2  $n \leftarrow B.length$ 
3 Let  $Lev[0 \dots n]$ ,  $LevP[0 \dots n]$  be new arrays
4 for  $j = 0$  to  $n$  do
5    $Lev[j] \leftarrow j$ 
6 end
7  $LevP \leftarrow Lev$ 
8 for  $i = 1$  to  $m$  do
9   for  $j = 0$  to  $n$  do
10    if  $j = 0$  then
11       $Lev[j] \leftarrow i$ 
12    else if  $A[i - 1] = B[j - 1]$  then
13       $Lev[j] \leftarrow LevP[j - 1]$ 
14    else
15       $Lev[j] \leftarrow \min(LevP[j], LevP[j - 1], Lev[j - 1]) + 1$ 
16    end
17  end
18  if  $i \neq n$  then
19     $Swap(Lev, LevP)$ 
20  end
21 end
Output:  $Lev[n]$ 

```

size on a given machine. This algorithm is later modified to compute the edit distance and is presented in [10]. In [17], Myers bit parallel algorithm [16] is also implemented on GPUs using collaborative parallelization for large bitwise operations and concurrent pattern matching. An implementation of Myers algorithm [16] is also presented in [18].

An obvious way to compute the edit distance is to use diagonal parallel approach. It calculates the edit distance table diagonal-wise by simultaneously computing all entries in a diagonal. Its main disadvantage is unbalanced workload among processors because sizes of the diagonals vary in each step [19]. In [20], an efficient parallel algorithm for longest common sub-sequence problem is presented for shared memory multi-core systems and GPUs. Sadiq et al. [21] presented a parallel algorithm for the edit distance problem. Authors resolved the dependences in the dynamic programming table and manage to calculate the edit distance table row-wise where every row is computed in parallel. An additional preprocessing step is added which helps in resolving the dependences in the dynamic programming table. Their proposed algorithm is evaluated on GPUs and multi-core systems. It achieved good

		C	A	R	E	
i, j		0	1	2	3	4
	0	0	1	2	3	4
A	1	1	1	1	2	3
C	2	2	1	1	2	3
E	3	3	2	2	3	2
R	4	4	3	3	2	3

Figure 1. Edit distance table between strings: “ACER” and “CARE” and dependences of a cell (i, j) in the edit distance table

performance gains. Similar strategies to [21] for the problem of approximate string matching have been proposed in [22, 23].

Another problem that is related to the edit distance is the sequence alignment problem. In that, similar regions of two sequences are aligned together by inserting gaps in the sequences. Major algorithms of the sequence alignment problem also follow the dynamic programming. Furthermore, their solutions have similar dependences in the dynamic programming table as in the case of edit distance. Comprehensive studies have been made for this problem. Aluru et al. [24] presented a distributed algorithm for various algorithms of biological sequence comparison. It introduces the way to calculate the alignment table row-wise or column-wise by using parallel scan operation [12, 25]. This algorithm is implemented by using MPI. Results are presented for two type of sequences: one having complete match case and other having complete mismatch case. This algorithm achieved good speedup on both setups. Scalability of their algorithm is evaluated by a varying number of processors. Authors indicated that their method can be used to parallelize other algorithms that needed to calculate such a score table. An exact parallel space and time optimal algorithm for the sequence alignment is proposed in [26]. It also uses the parallel scan operation to exploit parallelism. Furthermore, it is important to note that edit distance can also be computed using the parallel scan operation [25].

In [27] two streaming algorithms for biological sequence alignment are presented for GPUs. These streaming algorithms are also based on diagonal parallel approach. A parallel algorithm for local alignment is presented in [28]. It uses a master and slave model. This algorithm is also space efficient. It is evaluated by using MPI and

cluster of eight and sixty nodes. The authors evaluated their algorithm by using random pair of sequences in the range of 1 KBP (Kilo Base Pairs) and 1 600 KBP.

A parallel and space efficient algorithm for sequence alignment called *z-align* is presented in [29]. The authors executed their algorithm in four phases:

1. distribution of input data including sequences,
2. calculation of the similarity matrix,
3. gathering of the best score and their coordinates at the master processor,
4. obtaining actual alignment(s) in limited space using a master-slave model and self-scheduling policy.

This algorithm is evaluated on a cluster of sixteen processors. The authors compared sequences of size between 1 KB and 3 MB. A parallel algorithm for multiple sequence alignment is proposed in [30]. It is evaluated using a cluster of systems connected through network. Parallelism is achieved by partitioning the dynamic programming matrix among host systems. The authors also evaluated the scalability of the algorithm.

To summarize, the following are major principles on which parallel computation of the edit distance has been done. Some earlier solutions are based on a parallel random access machine model that is really fast in terms of time complexity [8, 9, 13], but due to the quadratic space complexity and resource requirements such models are not implemented practically. Bit parallel approach [10, 16, 17] is quite common, in which a couple of cells are represented as a single word and bitwise operations are used to perform simultaneous computations. Another approach that is quite common is diagonal based approach [19] in which edit distance table is solved anti-diagonal-wise, while computing each anti-diagonal in parallel. There is not dependence among the cells of anti-diagonal according to the algorithm presented as Algorithm 1. It can be further seen in the Figure 1. The number of cells in each anti-diagonal is different, what lowers the parallelism in certain stages of the algorithm. Furthermore, it is also not a load balanced approach. Another way of solving the edit distance table is to use the parallel scan approach, which resolves the dependences in the table and computes it row-wise. It increases the number of steps for computation of each row, but it is the load balanced approach. Another method of resolving the dependences in the dynamic programming table involves a preprocessing step [21, 22, 23]. After changing the dependences, updated algorithm computes entries of each row of the table in parallel.

Overall, extensive studies have been made for parallel computation of the edit distance and related problems. Most of the studies primarily focus on GPU-based solution. In string comparison, problem size can be very large, therefore the scalable solution is required for these problems. A solution for distributed memory systems is always a good choice for scaling because there is no limit to the number of processing nodes. To the best of our knowledge, distributed solutions exist [24, 26, 28] in literature but those are not specific to the edit distance. Therefore, this study focuses on designing a distributed algorithm for parallel edit distance calculation.

5 A DISTRIBUTED ALGORITHM FOR PARALLEL EDIT DISTANCE COMPUTATION

To compute multiple entries of the edit distance table simultaneously, dependences for computing each cell should be investigated. Computations in the first row and the first column of the edit distance table are independent of any cell (Algorithm 1). For any cell (i, j) in i^{th} row, where $i > 0$ and $j > 0$, there are two possibilities: either cell (i, j) can be a *match case* where the character of string A matches with the corresponding character of string B . Otherwise, it will be a *non-match case*.

According to Algorithm 1, to compute any cell (i, j) which is a *non-match case*, three values should be known in advance: the value of left cell $(i, j - 1)$, the value of upper cell $(i - 1, j)$, and the value of diagonal cell $(i - 1, j - 1)$ (Figure 1). To compute any other cell in the edit distance table which is not a *non-match case*, only the values from $(i - 1)^{\text{st}}$ row are required.

Yousaf et al. [31] presented a parallel algorithm for the edit distance computation that resolves the dependences in the edit distance table. This algorithm computes all cells in the i^{th} row of the edit distance table simultaneously based on the $(i - 1)^{\text{st}}$ row only. Yousaf et al. [31] proved that the value of a *non-match case* can also be computed from $(i - 1)^{\text{st}}$ row with the following equation:

$$Lev[j] = \min(LevP[j] + 1, LevP[j - 1] + 1, LevP[mp - 1] + k).$$

Here mp is the position of the last *match case* in the i^{th} row and k is the distance of cell (i, j) from last *match case*. According to this new equation, dependences of the cell (i, j) of the edit distance table are changed (as illustrated in Figure 2). Based on these new dependences each cell (i, j) of an i^{th} row can be computed based on $(i - 1)^{\text{st}}$ row.

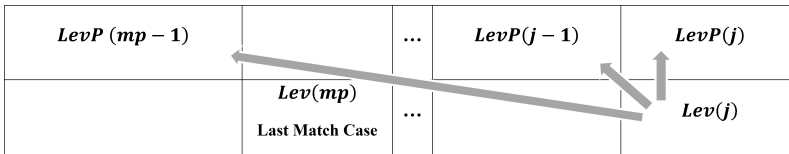


Figure 2. Dependences of cell (i, j) according to parallel edit distance algorithm presented in [31]

Given character set Σ having $|\Sigma|$ number of unique characters, last *match case* for each cell in an i^{th} row can be found by computing a Last Match Case Table (*LMT*) of size $(|\Sigma| - 1) \times n$. It contains the last match positions of the unique characters of character set Σ against string B . It can be computed by using Algorithm 2.

Table 1 shows a sample *LMT* for character set $\{A, C, G, T\}$ and String = “ABACUS”.

Algorithm 2: Computing LMT

Input: Character set Σ and String B

```

1  $|\Sigma| \leftarrow \Sigma.length$ 
2  $n \leftarrow B.length$ 
3 Let  $LMT[0 \dots |\Sigma| - 1, 0 \dots n]$  be a new table
4 for  $i = 0$  to  $|\Sigma| - 1$  do
5   for  $j = 0$  to  $n$  do
6     if  $j = 0$  then
7        $LMT[i][j] \leftarrow 0$ 
8     else if match case then
9        $LMT[i][j] \leftarrow j$ 
10    else
11       $LMT[i][j] \leftarrow LMT[i][j - 1]$ 
12    end
13  end
14 end

```

	0	1	2	3	4	5	6
		A	B	A	C	U	S
A	0	1	1	3	3	3	3
C	0	0	0	0	4	4	4
G	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0

Table 1. LMT for $\Sigma = \{A, C, G, T\}$ and String = "ABACUS"

The value of the last match position can be incorporated with the following equation:

$$Lev[j] = \min(LevP[j] + 1, LevP[j - 1] + 1, LevP[LMT[c][j] - 1] + (j - LMT[c][j])).$$

Here $LMT[c][j]$ is the position of the last *match case* in an i^{th} row. This algorithm computes the same edit distance score as sequential algorithm. Its proof of correctness is established in [31] and [21]. Furthermore, the size of each row is the same in the edit distance table, therefore this approach allows balanced work division among the processing elements.

Now, let us introduce a distributed way of computing edit distance based on [31].

5.1 Distributed Algorithm

The proposed method is divided into three parts:

1. Distribution of the strings,

2. Distributed computation of the *LMT*,
3. Distributed computation of edit distance table using *LMT*.

Assume that there are p processors having ID in the range of 0 to $p - 1$. For simplicity, also assume that n (size of string B) is divisible by p . However, it can be generalized for an arbitrary number of processors. Now, we will discuss all three steps one by one.

5.1.1 Distribution of the Strings

Initially, strings are distributed among each computing machine as plain text files. Only a respective part of the string is moved to main memory on which processing is required. Edit distance table can be computed row by row by reformulating the dependences (as shown in [21, 31]). Each row can be divided among multiple processors. So, every processor will compute $O\left(\frac{n}{p}\right)$ part of the row. To compute its part of the row each processor needs $O\left(\frac{n}{p}\right)$ fraction of the string B , therefore each processor p_r gets second string from $B[r(\frac{n}{p})]$ to $B[(r + 1)\frac{n}{p}]$ where r is ID of the processor. This process is illustrated in Figure 3 which shows the distribution of a row and string B among p processors.

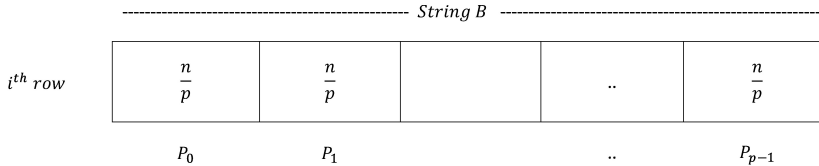


Figure 3. Distribution of a row of the edit distance table and string B among p processors

In the edit distance table, computing a row requires only one character from string A . So, string A can be obtained in chunks by each processor. Every time processing on one chunk is completed, next chunk is obtained.

5.1.2 Distributed Computation of the *LMT*

The computation of the *LMT* is divided in equal parts of $\frac{n|\Sigma|}{p}$ among the p processors (as illustrated in Figure 4).

LMT is computed row by row by modifying Algorithm 2. Each processor can compute its part of i^{th} row of the *LMT* in two steps:

1. In the chunk, all values after first *match case* can be computed in a similar manner to Algorithm 2 and all the values before first *match case* are undefined initially.

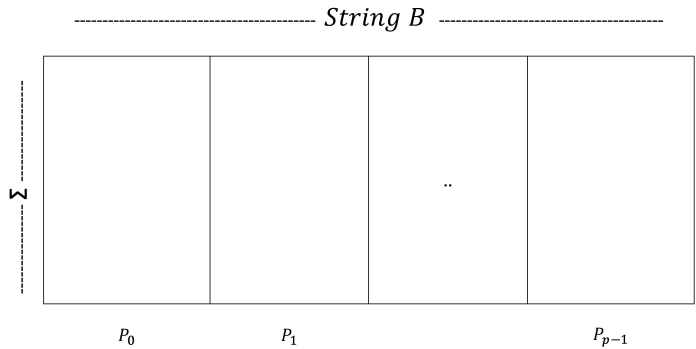


Figure 4. Distribution of the *LMT* among p processors

- 2. In any row of the *LMT*, values always increase or remain the same. So, all the values before first *match case* can be found by an exclusive scan operation (with maximum as binary associative operator) on extreme right value in the chunk of each processor. This value can be used in the place of undefined values.

Hence with these steps, each processor has all required values of i^{th} row of the *LMT*. Now consider an example of computation of the *LMT* where a row of the table is computed for a character ‘A’ and string “ATACG”. The whole row is divided among four processors in $\frac{n}{p}$ chunks where the size of each chunk is two. Table 2 illustrates both the steps to compute a row of the *LMT*.

	0	1	2	3	4	5
	$P_0(0)$	$P_0(1)$	$P_1(0)$	$P_1(1)$	$P_2(0)$	$P_2(1)$
(A)		A	T	A	C	G
Values computed after step 1	0	1	–	3	–	–
Values received after step 2	–		1		3	

Table 2. Steps to compute *LMT* (for a character ‘A’ and string “ATACG”)

Similarly, all rows of the *LMT* can be computed. At one time, all the processors will be computing their chunk of one specific row. After computation of a row, all processors are synchronized. Then next row is computed. The procedure to compute the *LMT* for an arbitrary process p_r is presented as Algorithm 3. Algorithm 3 takes string B and character set as input and computes the *LMT*.

5.1.3 Distributed Computation of the Edit Distance Table

The computation of the edit distance table is also divided in equal parts of $\frac{mn}{p}$ among p processors (as illustrated in Figure 5).

Each processor will compute $\frac{mn}{p}$ part of the edit distance table. Edit distance table is computed row by row, therefore to store current and previous row of the edit

Algorithm 3: Computation of the *LMT* at processor p_r where r is a unique identifier for the process between $[0 \dots p-1]$

Input: Character set Σ and String B

```

1   $|\Sigma| \leftarrow \Sigma.length$ 
2   $n \leftarrow B.length$ 
3   $B_r \leftarrow B[r \frac{n}{p} \dots r(\frac{n}{p} + 1)]$ 
4  Let  $LMT[0 \dots (|\Sigma| - 1), 0 \dots \frac{n}{p}]$  be a new table
5  for  $i = 0$  to  $|\Sigma| - 1$  do
6       $jc \leftarrow$  initial index of a row in  $p_r$ 's chunk
7      for  $j = 0$  to  $\frac{n}{p}$  do
8          if  $jc = 0$  then
9               $LMT[i][j] \leftarrow 0$ 
10         else if  $B_r[j] \neq \Sigma[i]$  and  $j = 0$  then
11              $LMT[i][j] \leftarrow undef$ 
12         else if  $B_r[j] == \Sigma[i]$  then
13              $LMT[i][j] \leftarrow jc$ 
14         else
15              $LMT[i][j] \leftarrow LMT[i][j - 1]$ 
16         end
17          $jc++$ 
18     end
19     Values before first match case  $\leftarrow$  Exclusive Scan (Max) on
         $LMT[|\Sigma| - 1][\frac{n}{p} - 1]$ 
20 end
```

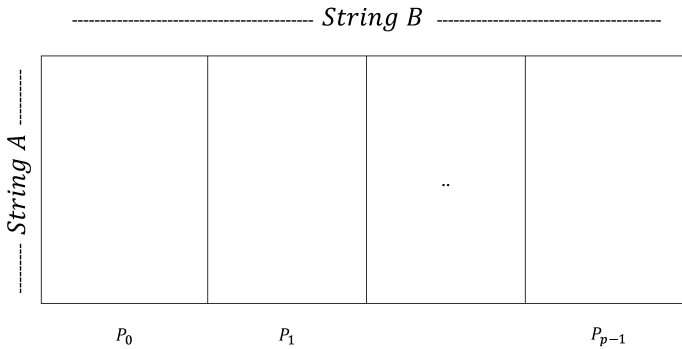


Figure 5. Distribution of the edit distance table among p processors

distance table, each process will allocate $O\left(\frac{n}{p}\right)$ space for both rows. At one time, all processors will be computing their part of one specific row. After computation of the row, all processors are synchronized. Then the next row is computed, and the same process follows. Computation of an i^{th} row of the edit distance table is done in similar manner to Algorithm 1 but the following two cases must be dealt with.

1. To compute the extreme left value in chunk of each processor, the value of its diagonal cell is required (Figure 2), which is not available locally. It can be found from the preceding processor.
2. It is possible that for some initial cells in the chunk of a processor, the value of the last *match case* lies in the chunk of the preceding processor/s (Figure 2). Therefore, that value must be obtained before computation of the i^{th} row.

These both cases can be managed by two communication steps. The first case can be handled as follows: Before computation of an i^{th} row, every p^{th} processor communicates its extreme right value of the $(i - 1)^{\text{st}}$ row with $(p + 1)^{\text{st}}$ processor. To handle the second case, an exclusive scan operation (with maximum as binary associative operator) is performed with the value at very last *match case* in each processor's chunk (as there can be more than one *match cases* in each processor's chunk).

Now, each processor has every value to compute its part in i^{th} row of the edit distance table. Furthermore, all the values in the chunk of each processor can also be computed in parallel as their computation is dependent only on $(i - 1)^{\text{st}}$ row [31]. In multi-core system, we can divide this computation further among multiple threads. Similarly, all the rows can be computed. The procedure to compute the edit distance table is presented as Algorithm 4 which shows working of an arbitrary process p_r . This algorithm takes strings A and B as input and computes the value of the edit distance.

5.2 Analysis of the Algorithm

5.2.1 Computational Complexity

LMT and the edit distance table are computed row by row and each row is equally divided among processors. Therefore, each processor computes $O\left(\frac{n|\Sigma|}{p}\right)$ part of the LMT and $O\left(\frac{mn}{p}\right)$ part of the edit distance table. So, total computational complexity is $O\left(\frac{mn}{p}\right)$ because $\frac{n|\Sigma|}{p} < \frac{mn}{p}$. Edit distance table is further divided among the available threads in case of hybrid implementation. If number of available threads is t , then computation time is reduced to $O\left(\frac{mn}{pt}\right)$ because chunk of row for each process is further divided among the cores of a multi-core system. Hence, the computation of a row would take $O\left(\frac{n}{pt}\right)$ time.

Algorithm 4: Computation of edit distance table at processor p_r

Input: Strings: A and B , and LMT

```

1  $m \leftarrow A.length$ 
2  $n \leftarrow B.length$ 
3  $B_r \leftarrow B[r \frac{n}{p} \dots r(\frac{n}{p} + 1)]$ 
4  $Ac \leftarrow A[0 \dots \frac{n}{p}]$ 
5 Let  $Lev[0 \dots \frac{n}{p}]$ ,  $LevP[0 \dots \frac{n}{p}]$  be new arrays
6  $j\_initial \leftarrow$  initial index of a row in  $p_r$ 's chunk
7  $jc \leftarrow j\_initial$ 
8 for  $j = 0$  to  $\frac{n}{p} - 1$  do
9    $Lev[j] \leftarrow jc$ 
10   $jc++$ 
11 end
12  $LevP \leftarrow Lev$ 
13 for  $i = 1$  to  $m$  do
14   Get next chunk of  $A$  if required
15    $ch \leftarrow$  next character in  $A$ 
16    $jc \leftarrow j\_initial$ 
17   for  $j = 0$  to  $\frac{n}{p} - 1$  do
18     if  $jc = 0$  then
19        $Lev[j] \leftarrow i$ 
20     else if  $Ac[i] = B_r[j]$  then
21       if  $j == 0$  then
22          $Lev[j] \leftarrow pre\_end\_value$ 
23       else
24          $Lev[j] \leftarrow LevP[j - 1]$ 
25       end
26     else
27        $c \leftarrow$  row index of character  $Ac[i]$  in  $LMT$ 
28        $lmp \leftarrow LMT[c][j]$ 
29        $lmv \leftarrow$  Value at last match case according to  $lmp$ 
30        $Lev[j] \leftarrow \min(Lev[j] + 1, LevP[j - 1] + 1, (jc - lmp) + lmv)$ 
31     end
32      $jc++$ 
33   end
34   if  $i \neq n$  then
35     if  $r \neq p - 1$  then
36        $end\_value \leftarrow Lev[\frac{n}{p} - 1]$ 
37       Send  $end\_value$  to processor  $p_{r+1}$ 
38     end
39     if  $r \neq 0$  then
40       Receive  $pre\_end\_value$  from processor  $p_{r-1}$ 
41     end
42      $lmv \leftarrow$  Exclusive Scan (Max) on very last match case in  $p_r$ 's chunk
43     Swap( $Lev$ ,  $LevP$ )
44   end
45 end
Output:  $Lev[\frac{n}{p} - 1]$  if  $(r == p - 1)$ 

```

5.2.2 Communication Time

Total communication required for the *LMT* is $|\Sigma|(t_s + t_w\eta) \log n$ because to compute one row of the *LMT*, one exclusive scan operation is required. In the edit distance table, one exclusive scan operation per row is required. Furthermore, for each row one extreme right value in the chunk of the processor should also be communicated. Its communication time is $m(t_s + t_w\eta)(\log n + 1)$. So, the total communication time for this algorithm is $|\Sigma|(t_s + t_w\eta) \log n + m(t_s + t_w\eta)(\log n + 1)$.

5.2.3 Space Complexity

Space requirement for string B is $O\left(\frac{n}{p}\right)$. *LMT* requires $O\left(\frac{n|\Sigma|}{p}\right)$ space. Edit distance table requires $O\left(\frac{n}{p}\right)$ space. String A can be obtained by processors in arbitrary sized multiple chunks, but if the chunk size taken is less than $O\left(\frac{n}{p}\right)$ then space complexity would be optimal. Hence, the total space complexity would be $O\left(\frac{n}{p}\right) + O\left(\frac{n|\Sigma|}{p}\right)$. Here $|\Sigma|$ is constant. So, the overall space complexity is $O\left(\frac{n}{p}\right)$.

5.2.4 Comparison with the Existing Algorithms

Table 3 shows the time and space complexity of algorithms that are closely related to the edit distance. Although some of them are not proposed for the distributed memory environment, this comparison suggests that our proposed algorithm is equally efficient as most of the state-of-the-art algorithms in terms of time and space complexity.

Algorithm	Time Complexity	Space Complexity
Huang [14]	$\frac{(m+n)^2}{p}$	$\frac{m+n}{p}$
Myers [16], Chacón et al. [17]	$\frac{mn}{w}$	mn
Šošić and Šikić [18]	$\frac{mn}{w}$	$m + n$
Sadiq et al. [21]	$\frac{mn}{p}$	$m + n$
Aluru et al. [24]	$\frac{mn}{p}$	$m + \frac{n}{p}$
Rajko and Aluru [26]	$\frac{mn}{p}$	$\frac{m+n}{p}$

Table 3. Space and time complexity of the existing algorithms related to the edit distance

Here m and n are lengths of string A and string B , respectively, w is the maximum word size that a machine can process, and p is the number of processing units. Faster algorithms [9, 13] are proposed for the parallel random access machine (PRAM) model but they are never implemented practically. Their space complexity is also quadratic.

6 EXPERIMENTS AND RESULTS

We have used five-nodes cluster with a minimum specification of one node: Intel Core-i5-3570K 3.40 GHz CPU having 4 physical cores, 4 logical processors, and 8 GB of main memory. All nodes in the cluster are interconnected to centralized hub by using fast ethernet cables. Furthermore, we used MPI (mpich version 3.2) for implementation of the algorithms. Four processes are launched at each node of the cluster for the pure MPI implementation (twenty processes in total for all nodes in the cluster). We have also used OpenMP for parallelism on one node. In the OpenMP implementation, computation inside the chunk of a row of each process is divided among multiple threads. Those threads are mapped into multiple cores. We have used OpenMP pragmas for static division of the work among the multiple threads. In the hybrid experiments using MPI + OpenMP, total five processes are launched (one for the each node of the cluster), where each process launches four threads to completely utilize all the cores of one node in the cluster.

We compared the proposed algorithm with an existing parallel scan approach [24] that involves a higher number of steps than the proposed algorithm. Experiments are performed for two types of datasets:

- Random strings
- Real DNA strings obtained from the National Center for Biotechnology Information website (NCBI) [32]

6.1 Experiments with Random Strings

In the first set of experiments, we have used strings in the range of 100 000 to 1 000 000. Strings are generated randomly from twenty-six letters in the Latin alphabets. In each experiment, strings of equal size are compared.

6.1.1 Distributed Memory Setup (MPI)

Results for the MPI by using randomly generated strings show that the proposed algorithm achieved speedup up to $5.90\times$ and the existing parallel scan approach [24] achieved speedup up to $4.33\times$. It is evident from the results that with increase in the problem size, the performance gain of the proposed algorithm is larger than the existing parallel scan approach. Figure 6 shows the scaled execution time and speedup for different sizes of problem with randomly generated strings.

6.1.2 Hybrid Setup (MPI + OpenMP)

Results show that the proposed algorithm achieved speedup up to $9.93\times$ and the existing parallel scan approach [24] achieved speedup up to $7.04\times$. Figure 7 shows the results for hybrid setup of MPI and OpenMP.

Results show that by using hybrid solution maximum attained speedup of the proposed algorithm is improved from $5.90\times$ to $9.93\times$. In the existing parallel scan

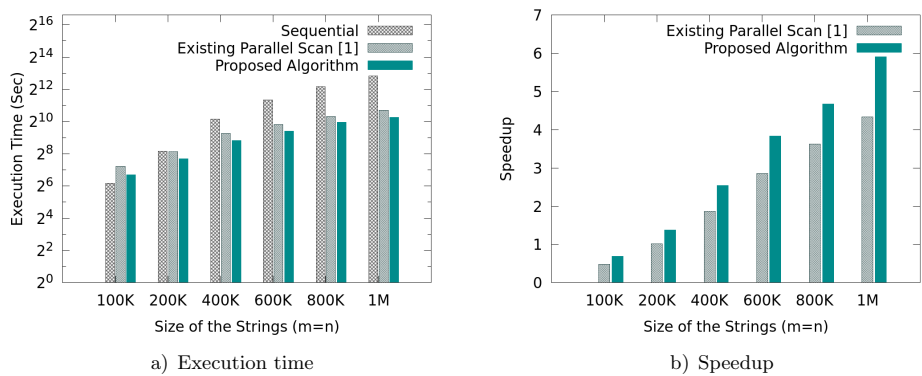


Figure 6. Results and comparisons for MPI with randomly generated strings

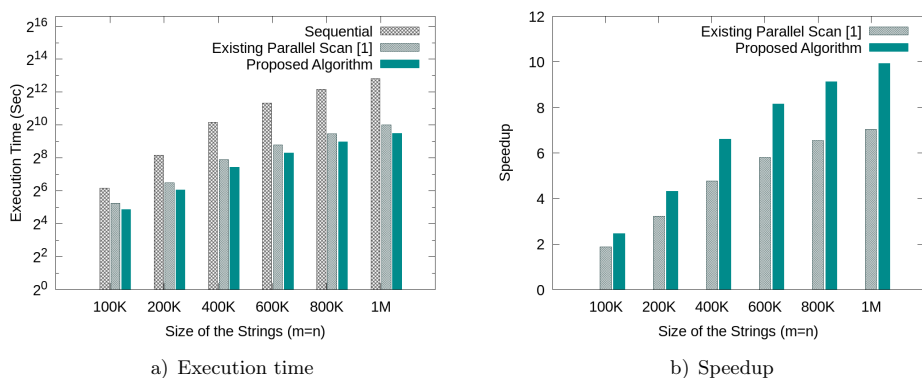


Figure 7. Results and comparisons for hybrid (MPI + OpenMP) setup with randomly generated strings

approach, maximum attained speedup is improved from $4.33\times$ to $7.04\times$. Hence, in the MPI-only implementation the performance is slower because explicit inter-node communication is required among the processes running on one node of the cluster.

6.2 Experiments with DNA Strings

In the next set of experiments real DNA strings are also compared. DNA strings are taken from NCBI website [32] which maintains a database of many DNA strings. Information of the DNA strings which are compared is presented in Table 4. This table also shows the value of edit distance between each two DNA strings.

Size of the *LMT* is reduced in DNA strings because the number of characters in character set of the DNA strings is four ($\{A, C, G, T\}$) which is smaller than the

Experiment ID	String A		String B		Edit Distance
	Name	Size	Name	Size	
Exp1	gbgss201	156 931	gbpln104	79 314	91 590
Exp2	gbgss201	156 931	gbhtg11	606 452	450 982
Exp3	gbhtg11	606 452	gbgss116	1 517 819	969 770
Exp4	gbuna1	308 453	gbinv32	3 424 429	3 116 517

Table 4. The list of experiments for DNA strings comparison

character set of randomly generated strings of latin letters. Hence, in this case, less time is required for the processing of the *LMT*.

6.2.1 Distributed Memory Setup (MPI)

Results on the MPI show that the proposed algorithm achieved speedup up to $11.05\times$ and the existing parallel scan approach [24] achieved speedup up to $5.44\times$.

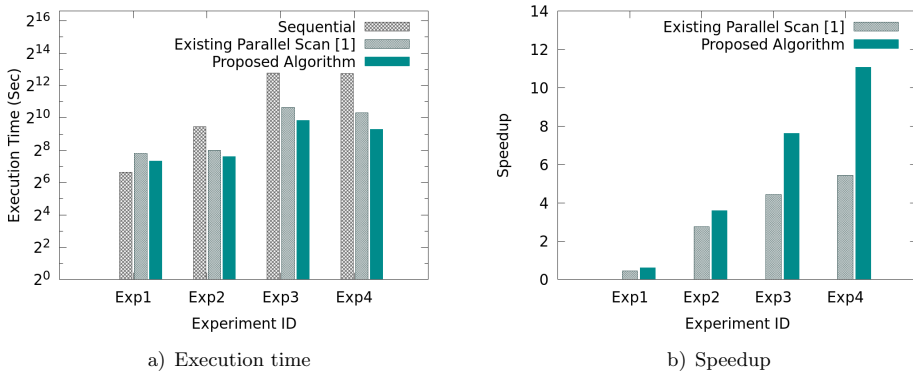


Figure 8. Results and comparisons for MPI-only setup with DNA strings

Figure 8 shows scaled execution time and speedup for the experiments mentioned in Table 4 for MPI.

6.2.2 Hybrid Setup (MPI + OpenMP)

Results on hybrid setup of MPI and OpenMP show that the proposed algorithm achieved speedup up to $12.49\times$ and the existing parallel scan approach [24] achieved speedup up to $5.68\times$. Speedup is improved compared to the MPI-only results (where maximum obtained speed up is $11.05\times$ and $5.44\times$ for the proposed algorithm and the existing parallel scan approach, respectively). In the MPI-only implementation, explicit inter-node and intra-node communication is required, while in the hybrid implementation, explicit inter-node communication is avoided. Performance gain for the proposed algorithm (by using OpenMP with MPI) is larger compared to the

existing parallel scan approach where proposed method gained added speedup of up to $1.43\times$ and the existing parallel scan method boosted up to 0.24 times. Hence, parallel scan method requires more communication than the proposed algorithm. Figure 9 shows the experimental results on hybrid setup of MPI and OpenMP by using DNA strings.

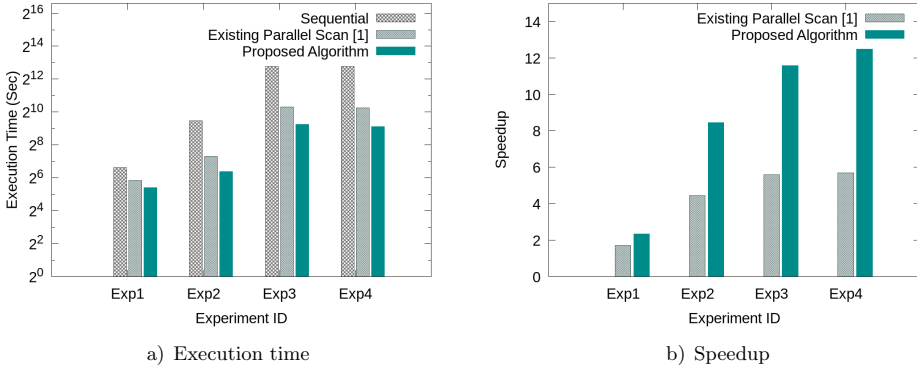


Figure 9. Results and comparisons for hybrid (MPI + OpenMP) setup with DNA strings

Since the computation as well as communication time plays its role in the overall running time of the algorithm, both are equally important. Furthermore, in the proposed method an additional table (*LMT*) is also computed, therefore it is essential to analyze its running time as well. In the Table 5 and Table 6, detailed communication and computation time is presented (for the proposed algorithm) for the *LMT* and edit distance table. Results show that time required for the processing of the *LMT* is negligible compared to the edit distance table. Overall communication time is less than computation time and total running time increases with increase in problem size. Furthermore, it can be observed that communication time for MPI-only experiments is significantly larger than for the hybrid experiments due to added explicit inter-node communication among the processes in case of MPI-only implementation.

An interesting case can be seen in the results of Exp3 and Exp4. For the proposed algorithm, running time of Exp4 is smaller than Exp3 despite the size of the edit distance table being approximately the same in both cases (9.20×10^{11} and 1.05×10^{12} respectively for Exp3 and Exp4). This is due to the fact that in the approximately same sized problems, overall running time would be smaller for the tables having small number of large sized rows (where the size of string *A* determines the number of rows and the size of string *B* determines the size of a row). Exp4 has larger sized rows than Exp3, therefore in the one row more parallelism is achieved. It also has a small number of rows which reduces communication and synchronization overhead required after the computation of a row. This observation indicates that running time is small if the computation required is larger than communication,

i.e., the size of a row is large and the number of the rows is small. This indicates that communication and synchronization are more expensive operations than the computation.

Exp. ID	LMT		Edit Distance Table		Total		
	Comp.	Comm.	Comp.	Comm.	Comp.	Comm.	All
Exp1	0.006	0.015	67.437	93.214	67.444	93.229	160.673
Exp2	0.003	0.021	92.653	103.947	92.656	103.967	196.623
Exp3	0.004	0.023	493.335	426.950	493.339	426.973	920.311
Exp4	0.005	0.021	411.978	212.800	411.983	212.822	624.804

Table 5. Detailed communication and computation time (in seconds) for the proposed algorithm using MPI-only implementation

Exp. ID	LMT		Edit Distance Table		Total		
	Comp.	Comm.	Comp.	Comm.	Comp.	Comm.	All
Exp1	0.002	0.002	20.054	22.389	20.056	22.391	42.447
Exp2	0.003	0.001	67.764	15.678	67.767	15.678	83.446
Exp3	0.007	0.002	534.848	71.853	534.856	71.855	606.711
Exp4	0.015	0.001	505.576	47.455	505.591	47.456	553.047

Table 6. Detailed communication and computation time (in seconds) for the proposed algorithm using hybrid setup of MPI and OpenMP

6.3 Summary of the Results

These results suggest that the proposed algorithm significantly outperforms the existing parallel scan approach [24]. It is up to $12.49\times$ faster than a sequential algorithm. It also utilizes given resources effectively compared to the existing parallel scan approach. For the small problem sizes, the sequential algorithm performs better compared to the parallel solutions because of the communication and synchronization overheads of parallel solutions. In the proposed algorithm, time required for the processing of the *LMT* is negligible compared to the edit distance table. Size of the *LMT* is smaller for the DNA strings compared to randomly generated strings (as there are only four characters in character set of DNA strings), therefore processing time of the *LMT* is smaller for DNA strings. Furthermore, in the proposed approach, the overall communication time is smaller compared to the computation time which is always desired because the communication is expensive compared to the computation.

Another important factor that should be considered is parallel efficiency of both algorithms. It is the ratio of speedup and number of compute nodes. Total compute nodes are twenty. Maximum speedup attained for the proposed algorithm is $12.49\times$ and $7.04\times$ for parallel scan approach. Hence, maximum parallel efficiency of the proposed algorithm is 0.62 and it is 0.35 for parallel scan approach.

The reason for this low parallel efficiency is that the explicit synchronization and communication is required at the end of computation of each row. Due to these overheads the parallel efficiency is decreased. In case of the parallel scan approach, parallel efficiency is even lower because it involves more steps which add more parallel overheads. These overheads are unavoidable because of the nature of the algorithm. Moreover, in general, these trade-offs exist when designing the parallel algorithm.

7 CONCLUSION AND FUTURE WORK

In this paper, we introduced a distributed algorithm for the parallel edit distance computation. The proposed algorithm ensures a balanced workload among the processors. To the best of our knowledge, this is the first time when a distributed algorithm of the edit distance is presented. Earlier studies [14, 24] have a mention about distributed computation of edit distance, but its practical implementation is not proposed.

We have presented results for random and real DNA strings. Evaluation on hybrid setup of OpenMP and MPI shows that the proposed algorithm achieved $12.49\times$ speedup compared to the sequential version of the algorithm. Furthermore, it also outperforms the parallel scan approach [24] significantly. There are many other problems that are related to edit distance. Damerau–Levenshtein distance, approximate string matching, longest common sub-sequence, and sequence alignments are an example of such problems. It is good challenge to design their solutions for distributed memory systems. Furthermore, in future we intend to design a distributed memory solution that could exploit parallelism by using GPU on its each node.

REFERENCES

- [1] DROPPA, J.—ACERO, A.: Context Dependent Phonetic String Edit Distance for Automatic Speech Recognition. 2010 IEEE International Conference on Acoustics, Speech, and Signal Processing, Dallas, Texas, USA, 2010, pp. 4358–4361, doi: 10.1109/ICASSP.2010.5495652.
- [2] LI, H.—HOMER, N.: A Survey of Sequence Alignment Algorithms for Next-Generation Sequencing. *Briefings in Bioinformatics*, Vol. 11, 2010, No. 5, pp. 473–483, doi: 10.1093/bib/bbq015.
- [3] VAROL, C.—ABDULHADI, H. M. T.: Comparision of String Matching Algorithms on Spam Email Detection. 2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT), IEEE, Ankara, Turkey, 2018, pp. 6–11, doi: 10.1109/IBIGDELFT.2018.8625317.
- [4] YING, Z.—ROBERTAZZI, T. G.: Signature Searching in a Networked Collection of Files. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, 2014, No. 5, pp. 1339–1348, doi: 10.1109/TPDS.2013.258.

- [5] YU, M.—LI, G.—DENG, D.—FENG, J.: String Similarity Search and Join: A Survey. *Frontiers of Computer Science*, Vol. 10, 2016, No. 3, pp. 399–417, doi: 10.1007/s11704-015-5900-5.
- [6] NAVARRO, G.: A Guided Tour to Approximate String Matching. *ACM Computing Surveys (CSUR)*, Vol. 33, 2001, No. 1, pp. 31–88, doi: 10.1145/375360.375365.
- [7] WAGNER, R. A.—FISCHER, M. J.: The String-to-String Correction Problem. *Journal of the ACM (JACM)*, Vol. 21, 1974, No. 1, pp. 168–173, doi: 10.1145/321796.321811.
- [8] MASEK, W. J.—PATERSON, M. S.: A Faster Algorithm Computing String Edit Distances. *Journal of Computer and System Sciences*, Vol. 20, 1980, No. 1, pp. 18–31, doi: 10.1016/0022-0000(80)90002-1.
- [9] MATHIES, T. R.: A Fast Parallel Algorithm to Determine Edit Distance. Carnegie Mellon University. Journal contribution, 1988, doi: 10.1184/R1/6587387.v1.
- [10] HYYRÖ, H.: A Bit-Vector Algorithm for Computing Levenshtein and Damerau Edit Distances. *Nordic Journal of Computing*, Vol. 10, 2003, No. 1, pp. 29–39.
- [11] GROPP, W.—HOEFLE, T.—THAKUR, R.—LUSK, E.: *Using Advanced MPI: Modern Features of the Message-Passing Interface*. MIT Press, 2014.
- [12] GRAMA, A.—KUMAR, V.—GUPTA, A.—KARYPIS, G.: *Introduction to Parallel Computing*. Addison Wesley, 2003.
- [13] APOSTOLICO, A.—ATALLAH, M. J.—LARMORE, L. L.—MCFADDIN, S.: Efficient Parallel Algorithms for String Editing and Related Problems. *SIAM Journal on Computing*, Vol. 19, 1990, No. 5, pp. 968–988, doi: 10.1137/0219066.
- [14] HUANG, X.: A Space-Efficient Parallel Sequence Comparison Algorithm for a Message-Passing Multiprocessor. *International Journal of Parallel Programming*, Vol. 18, 1989, No. 3, pp. 223–239, doi: 10.1007/BF01407900.
- [15] SELLERS, P. H.: The Theory and Computation of Evolutionary Distances: Pattern Recognition. *Journal of Algorithms*, Vol. 1, 1980, No. 4, pp. 359–373, doi: 10.1016/0196-6774(80)90016-4.
- [16] MYERS, G.: A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming. *Journal of the ACM (JACM)*, Vol. 46, 1999, No. 3, pp. 395–415, doi: 10.1145/316542.316550.
- [17] CHACÓN, A.—MARCO-SOLA, S.—ESPINOSA, A.—RIBECA, P.—MOURE, J. C.: Thread-Cooperative, Bit-Parallel Computation of Levenshtein Distance on GPU. *Proceedings of the 28th ACM International Conference on Supercomputing (ICS '14)*, ACM, Munich, Germany, 2014, pp. 103–112, doi: 10.1145/2597652.2597677.
- [18] ŠOŠIĆ, M.—ŠIKIĆ, M.: Edlib: A C/C++ Library for Fast, Exact Sequence Alignment Using Edit Distance. *Bioinformatics*, Vol. 33, 2017, No. 9, pp. 1394–1395, doi: 10.1093/bioinformatics/btw753.
- [19] BALHAF, K.—SHEHAB, M. A.—AL-SARAYRAH, W. T.—AL-AYYOUB, M.—AL-SALEH, M.—JARARWEH, Y.: Using GPUs to Speed-Up Levenshtein Edit Distance Computation. *7th International Conference on Information and Communication Systems (ICICS)*, IEEE, Irbid, Jordan, 2016, pp. 80–84, doi: 10.1109/ICICS.2016.7476090.

- [20] YANG, J.—XU, Y.—SHANG, Y.: An Efficient Parallel Algorithm for Longest Common Subsequence Problem on GPUs. *Proceedings of the World Congress on Engineering (WCE 2010)*, London, U.K., 2010, Vol. 1, pp. 499–504.
- [21] SADIQ, M. U.—YOUSAF, M. M.—ASLAM, L.—ALEEM, M.—SARWAR, S.—JAFRY, S. W.: NvPD: Novel Parallel Edit Distance Algorithm, Correctness, and Performance Evaluation. *Cluster Computing*, Vol. 23, 2020, pp. 879–894, doi: 10.1007/s10586-019-02962-w.
- [22] HO, T.—OH, S.—KIM, H.: A Parallel Approximate String Matching Under Levenshtein Distance on Graphics Processing Units Using Warp-Shuffle Operations. *PloS ONE*, Vol. 12, 2017, No. 10, Art. No. e0186251, doi: 10.1371/journal.pone.0186251.
- [23] GUO, L.—DU, S.—REN, M.—LIU, Y.—LI, J.—HE, J.—TIAN, N.—LI, K.: Parallel Algorithm for Approximate String Matching with K Differences. *2013 IEEE Eighth International Conference on Networking, Architecture and Storage*, Xi'an, China, 2013, pp. 257–261, doi: 10.1109/NAS.2013.40.
- [24] ALURU, S.—FUTAMURA, N.—MEHROTRA, K.: Parallel Biological Sequence Comparison Using Prefix Computations. *Journal of Parallel and Distributed Computing*, Vol. 63, 2003, No. 3, pp. 264–272, doi: 10.1016/S0743-7315(03)00010-8.
- [25] SANDERS, P.—TRÄFF, J. L.: Parallel Prefix (Scan) Algorithms for MPI. In: Mohr, B., Träff, J. L., Worringer, J., Dongarra, J. (Eds.): *Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2006)*. Springer, Berlin, Heidelberg, *Lecture Notes in Computer Science*, Vol. 4192, 2006, pp. 49–57, doi: 10.1007/11846802_15.
- [26] RAJKO, S.—ALURU, S.: Space and Time Optimal Parallel Sequence Alignments. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, 2004, No. 12, pp. 1070–1081, doi: 10.1109/TPDS.2004.86.
- [27] LIU, W.—SCHMIDT, B.—VOSS, G.—MULLER-WITTIG, W.: Streaming Algorithms for Biological Sequence Alignment on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 18, 2007, No. 9, pp. 1270–1281, doi: 10.1109/TPDS.2007.1069.
- [28] BOUKERCHE, A.—DE MELO, A. C. M. A.—DE OLIVEIRA SANDES, E. F.—AYALA-RINCON, M.: An Exact Parallel Algorithm to Compare Very Long Biological Sequences in Clusters of Workstations. *Cluster Computing*, Vol. 10, 2007, No. 2, pp. 187–202, doi: 10.1007/s10586-007-0020-0.
- [29] BATISTA, R. B.—BOUKERCHE, A.—DE MELO, A. C. M. A.: A Parallel Strategy for Biological Sequence Alignment in Restricted Memory Space. *Journal of Parallel and Distributed Computing*, Vol. 68, 2008, No. 4, pp. 548–561, doi: 10.1016/j.jpdc.2007.08.007.
- [30] LOPES, H. S.—LIMA, C. R. E.—MORITZ, G. L.: A Parallel Algorithm for Large-Scale Multiple Sequence Alignment. *Computing and Informatics*, Vol. 29, 2012, No. 6+, pp. 1233–1250.
- [31] YOUSAF, M. M.—SADIQ, M. U.—ASLAM, L.—UL QOUNAIN, W.—SARWAR, S.: A Novel Parallel Algorithm for Edit Distance Computation. *Mehran University Research Journal of Engineering and Technology*, Vol. 37, 2018, No. 1, pp. 223–232, doi: 10.22581/muet1982.1801.20.

- [32] National Center for Biotechnology Information (NCBI), <https://www.ncbi.nlm.nih.gov/>.



Muhammad Umair SADIQ received his B.Sc. and M.Sc. degrees in computer science from PUCIT, University of the Punjab, Lahore, Pakistan in 2016 and 2018, respectively. His research interests include parallel and distributed computing, multi-core computing, performance analysis, and GPGPU computing.



Muhammad Murtaza YOUSAF is Professor at PUCIT, University of the Punjab, Lahore, Pakistan. He obtained his Ph.D. from University of Innsbruck, Austria in 2008. He worked on networks for grid computing during his Ph.D. His current areas of research include transport layer of networks, parallel and distributed computing, cloud computing, data science, and interdisciplinary research.

BREAST HISTOPATHOLOGY WITH HIGH-PERFORMANCE COMPUTING AND DEEP LEARNING

Mara GRAZIANI

*University of Applied Sciences of Western Switzerland
HES-SO Valais, Rue de Technopole 3
3960 Sierre, Switzerland*

&

*Department of Computer Science, University of Geneva
Battelle Building A, 7, Route de Drize
1227 Carouge, Switzerland
e-mail: mara.graziani@hevs.ch*

Ivan EGCEL

*University of Applied Sciences of Western Switzerland
HES-SO Valais, Rue de Technopole 3
3960 Sierre, Switzerland
e-mail: ivan.eggel@hevs.ch*

François DELIGAND

*INP-ENSEEIH
2 Rue Charles Camichel
31000, Toulouse, France
e-mail: francois.deligand@laposte.net*

Martin BOBÁK

*Institute of Informatics
Slovak Academy of Sciences
Dúbravská cesta 9, 845 07 Bratislava, Slovakia
e-mail: martin.bobak@savba.sk*

Vincent ANDREARCZYK

*University of Applied Sciences of Western Switzerland
HES-SO Valais, Rue de Technopole 3
3960 Sierre, Switzerland
e-mail: vincent.andrearczyk@hevs.ch*

Henning MÜLLER

*University of Applied Sciences of Western Switzerland
HES-SO Valais, Rue de Technopole 3
3960 Sierre, Switzerland
✉
Radiology Service, Medical Faculty, University of Geneva
Geneva, Switzerland
e-mail: henning.mueller@hevs.ch*

Abstract. The increasingly intensive collection of digitalized images of tumor tissue over the last decade made histopathology a demanding application in terms of computational and storage resources. With images containing billions of pixels, the need for optimizing and adapting histopathology to large-scale data analysis is compelling. This paper presents a modular pipeline with three independent layers for the detection of tumor regions in digital specimens of breast lymph nodes with deep learning models. Our pipeline can be deployed either on local machines or high-performance computing resources with a containerized approach. The need for expertise in high-performance computing is removed by the self-sufficient structure of Docker containers, whereas a large possibility for customization is left in terms of deep learning models and hyperparameters optimization. We show that by deploying the software layers in different infrastructures we optimize both the data preprocessing and the network training times, further increasing the scalability of the application to datasets of approximatively 43 million images. The code is open source and available on Github.

Keywords: Histopathology, exascale, medical imaging, sampling

1 INTRODUCTION

Breast cancer is the second leading cause of cancer death among women worldwide [35]. In 2019, the estimated number of women diagnosed with breast cancer was 271 270 only in the U.S. (7.3 % increase from the estimates of 2017), and an in-

creasing number of women died from the disease (2.8% increase from 2017 with 41 760 estimated deaths). A metastasizing breast cancer, particularly, has a far worse prognosis than a localized one. Assessing regional tumor spreading is thus extremely important for prompt treatment planning and accurate cancer staging [12]. Being the most likely target for initial metastases, axillary lymph nodes are analyzed to determine the spreading stage to neighboring areas. The N-stage of the TNM system, for instance, assesses the presence of tumor in regional lymph nodes. Before undergoing surgical removal of tissue, the patient is injected a blue dye or a radioactive tracer to identify the nearest lymph node to which the tumor may have drained, which is also called the sentinel lymph node [12]. Thin tissue slices are collected for visual analysis, mounted on glass slides. At this stage, the tissue slices mostly have transparent cells, the reason why they are treated with multiple contrasting stains. Different staining techniques can be used, with Hematoxylin and Eosin staining (H&E) being the most common, and immunohistochemical (IHC) staining for cytokeratin being used only in case of unclear diagnosis on H&E [7]. Hematoxylin stains the nuclei with blue, while Eosin highlights the cytoplasmic and non-nuclear components in different shades of pink. Their combination highlights the structure and cells within the tissue specimen. Tumor presence is traditionally evaluated by microscopic inspection, which may take several minutes per slide. Pathologists base their decisions upon morphometric and architectural features of the tissue structure and the nuclei, for instance estimating the presence of tubular formation, nuclear pleomorphism and mitotic count (see Figure 1) [30]. Such analysis is time-consuming, tedious, and error-prone since small metastases may be missed by the pathologist’s eyes (detection rate lower than 40% for the smallest type) [38]. Moreover, the grading of tumor growth patterns is very subjective and reports high inter-observer variability ($\kappa \in [0.582, 0.850]$ [32, 29]).

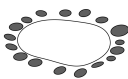




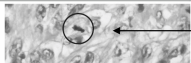
Clinical reference	Description	Visual model		Magnification
Degree of tubular formation [20]	tumour cells in gland structure	 well formed	 poorly formed	low
Nuclear pleomorphism	abnormality in size	 regular	 enlarged	high
	vesicular appearance		 uneven stain	
Mitotic count	number of mitosis	 mitosis		high

Figure 1. Grading criteria in the Nottingham Histologic Grade of breast cancer

The difficulties in the diagnostic motivate the automation of the grading process, made possible by the commercialization of high-resolution digital scanners for images of tissue specimens called Whole Slide Images (WSIs) around the year 2000 [1]. The well-established success of Convolutional Neural Networks (CNNs) in various

computer vision tasks triggered their use for automatic diagnoses. While clinical pipelines are increasingly introducing the full digitization of specimens by WSIs, several challenges affect the analysis of these images by deep learning models. On the data acquisition side, defects in the tissue handling, fixation, processing and staining affect the image quality, thus hampering future analyses. Intervals between tissue harvest, fixation and total fixation time are also poorly controlled in current pipelines, often leading to high variability and heterogeneity in the data collection [1]. Further technical aspects of the slide digitization such as the maximum magnification, the image compression and the color palette may also cause heterogeneity in the data, affecting both the image visualization and analysis. Moreover, storage requirements can easily explode when scaling the data collection to multiple patients. WSIs are extremely large, reaching generally more than $100\,000 \times 100\,000$ pixels [4]. Some pathological section processes may generate up to 20 images per patient, easily leading to more than 65 GB of data per patient [41]. Besides, no universally accepted WSI format has yet been defined, with different scanner manufacturers patenting different WSI file formats (such as SVS or NDPI). The multi-resolution pyramidal structure of WSIs, finally, contains multiple down-sampled versions of the original image, with different informative content [4]. Varying the scale at which the images are analyzed improves the quality of the analysis by taking into account information at different levels, the high-level disposition of the cells (e.g. degree of tubular formation) and fine-grain details such as the nuclei morphology or the mitotic activity.

This paper describes the research pipeline that we developed to process a scalable number of breast lymph node WSIs, to compare the performances of multiple deep learning architectures in terms of their training and inference time, accuracy and explainability. Our design takes into account the developmental requirements of scientific research generally performed on small computing clusters (up to 4 GPUs maximum). Being released with a containerized approach, it can be deployed on different infrastructures without requiring specific expertise, and it can be used to run some tasks on the large scale computing services provided by the PROCESS project¹, by transforming the Docker into Singularity containers. The design of such pipeline was driven by the need for flexibility in the used libraries for scientific development, while its integration exploits the services of the High-Performance Computing (HPC) structures. The paper is structured as follows. Section 2 introduces the state of the art about deep learning developments for both digital pathology and HPC. Section 3 describes the methods and datasets used for the experiments. In this section, we also describe the specific requirements of deep learning for histopathology and how these are met by the PROCESS platform interconnecting HPC and research centers in Europe. The experiments are run, in fact, at different sites, namely the University of Amsterdam (UVA), the SurfSARA computing center (with the LISA cluster), the SuperMUC-NG of the Leibniz-Rezerchcentrum (LRZ), the Prometheus cluster

¹ The PROCESS project received funding from the European Union's Horizon 2020. Homepage: <https://www.process-project.eu/>

at Cyfronet (AGH), the computing resources of the Slovak Academy of Sciences (UISAV) and our local resources. Particularly in Sections 3.5, 3.6 and 3.7, we describe the three layers of the proposed pipeline, pointing to the relative open source Github repositories. Section 4 reports the experimental results obtained with the proposed architecture on the different computing resources. In Section 5, finally, we present a discussion of the results and the challenges of scaling histopathology analysis to exascale datasets, together with the computational aspects of the software optimization.

2 RELATED WORK

2.1 Digital Pathology

Digital pathology has become very popular over the last decade for its practical assets [26]. WSIs reduce the need for storing glass slides on-site, reducing their risk of breaking, fading, or getting lost. Digital images can also easily be shared across institutions. This solicits the exchange of opinions, and if necessary pathology consults, about challenging cases. In case of specific questions at tumor boards, WSIs can be inspected offhand to find answers. Besides, WSIs allow the collection of examples of many different cases, including rare cases, that can be practically used in teaching slides sets. The creation of open-access digital slide archives, such as that of The Cancer Genome Atlas, led to intensive research on image analysis for pathology, which recently steered towards the application of deep learning based techniques. The large image sizes, as well as their heterogeneity and the often-imbalanced data distribution (towards non-tumorous tissue), make the development in this area challenging. The development of handcrafted features requires a background knowledge comprehensive of biology and imaging skills, together with the further specialization of the specific tissue type being analyzed. Handcrafted features that can be applied to any organ type include nuclear/gland shape and size, and tissue texture and architecture. Nuclear shape and texture, for example, were shown to predict the prognostics for breast cancer [25]. Graph-based approaches such as Voronoi tessellations were also used to characterize the relative positioning of nuclei and glands within the tissue. Prognostic features specific for breast cancer, for example, take into account the disposition of infiltrating tumoral lymphocytes.

Deep learning approaches for the analysis of histopathology images remove the need for the cumbersome creation of tissue-specific handcrafted features. Most of the approaches in the literature focus on the detection of Region of Interest (ROIs) where tumorous tissue can be identified, on the direct segmentation of nuclei [19] or the quantification of mitoses [3]. Because of the pyramidal structure of WSIs, images are often analyzed at multiple resolutions in a multi-step procedure [22, 42]. A thresholding operation (e.g. Otsu thresholding [42]) is performed on the lowest resolution image to filter out the image background. The doctor annotations of ROIs can then be used as ground-truth labels to train Convolu-

tional Neural Networks (CNNs). Image crops at higher resolutions are assigned the tumor label if they fall within the ROI while the remaining tissue is assigned a non-tumor label. These data are used to train CNNs end-to-end for the patch-based classification of tumorous images². Additionally to these, weakly supervised learning methods were used to exploit coarse labels to extract fine-grained information. The information contained in high-resolution crops of the WSIs is used as a “bag of inputs” with a single label in multiple-instance learning, for example. Teacher-student designs, besides, were recently proposed for combining small amounts of finely annotated data (manual segmentations of tumorous areas) with large amounts of weakly annotated data such as instance-level annotated WSIs [28].

As for the deployment of digital pathology on HPC, this is a recently growing area with yet unexplored opportunities and challenges. Particularly, with the exponential growth of biomedical data, several techniques will require adaptation and optimization to process efficiently large-scale datasets [41]. The specific demands of digital pathology require double adaptation, namely that of the HPC infrastructure towards the high computational burden of analyzing the massive dataset sizes, and that of developing pipelines that best exploit the infrastructure potential. Only a few existing works distribute the algorithms for medical image analysis. The work in [39], for example, uses MapReduce to answer spatial queries on large scale pathology images. A low-cost computing facility that can support the analysis of up to 1 million (1 M) images was proposed, for example, in [5]. Multiple-instance learning for histopathology was implemented in [41] to fit Microsoft HPC resources, and further modifications to the original algorithm were shown to obtain better scalability in [40]. These works, however, were directly optimized on the available computing resources for the research, requiring expertise at the frontier of HPC, deep learning and digital pathology. The main purpose of this work is to offer a ready-to-use application for researchers in the digital pathology field that have little to no experience in HPC and optimized computing. The application, being organized in three layers, presents different steps in the traditional pipeline with a modular approach, where each module can be customized in terms of research parameters and input data and can be deployed to different computing facilities.

2.2 HPC for Deep Learning

HPC infrastructures have played a fundamental part in solving large-scale problems with a high degree of computational complexity in domains such as astrophysics, high energy physics, computational fluid dynamics or finite element analysis. HPC services were built specifically to fit the requirements of such problems to efficiently scale up the problem size, exploiting the power of thousands of multi-core computer

² Different pipelines can be adopted, obtaining different results, as those in <https://camelyon17.grand-challenge.org/evaluation/leaderboard/>

nodes to effectively solve a single computationally-intensive problem. The downside is that the technical and organizational architecture of HPC trades some of the flexibility and dynamism for achieving the highest possible performance and maximum utilization rate. Only recently, HPC has embraced some of the programming solutions to provide more effective Single Instruction Multiple Data (SIMD) operations for vector data, which generated the possibility of introducing large scale machine learning applications to their computational capacities.

The training of a deep neural network involves solving a high-dimensional non-linear optimization function. The minimization of the gradients, often performed with gradient descent algorithms such as Stochastic Gradient Descent (SGD), uses several linear algebra computations on generally dense matrices. DistBelief was one of the first examples of parallel training frameworks to train fully connected networks (42 M parameters) and locally connected convolutional neural networks (17 M parameters) [9]. In this case, the parallelization of network training was achieved by *model parallelism*. Model parallelism, splits the weights of the network equally among multiple threads, creating network blocks that all work on the same mini-batch. In other words, the same data is used for every thread, but the model-inherent computations are split among threads. The output needs therefore to be synchronized after each layer to obtain the input to the next layer. A simpler method is that of *data parallelism*. The same model is replicated in each thread or working node (either GPU or CPU) and the training is performed on different batches of data. The gradients computed by each thread are relevant to the overall network training. Hence, they need to be shared within all the models on all the workers at the end of each data pass (i.e. their average is computed). Despite the simplicity and the adaptability of this method to datasets, models and resources, two possible bottlenecks may arise and hamper its efficiency. The averaging of the model parameters would require the transmission of extremely large matrices between nodes, thus requiring a highly efficient network card connecting each node. A similar problem arises when handling the multiple data accesses: the continuous input/output operations and data decoding may be the first cause for the slowing down of the training process. Moreover, as in data parallelism the same mini-batch is analyzed by all the GPUs, smaller batches result in decreased efficiency. The work on Large Minibatch SGD by [13] proposed a solution to the optimization difficulties that arise when increasing the size of the mini-batches to push to the edges the computational gains in each worker. Furthermore, recent research in gradient compression has proposed a solution to the latency in the communication between different nodes caused by limited communication bandwidth. For instance, gradient compression transmits only gradients larger than a certain threshold and accumulates locally the rest of the gradients, thus consistently reducing the network communication time [23].

In both data and model parallelism, the communication between the nodes can be either synchronous or asynchronous. In the former, all the devices use different parts of the same batch of training data and the model is updated when the computation has finished in all of them. In the latter, the devices update the

model independently based on their mini-batches. Generally, the updates are shared through a central parameter store, called parameter server. The parameter server receives the gradients from all the workers and, when all the updates have been received, computes the new model update and sends it to the workers. A boost in efficiency is given by the ring all-reduce technique, where each worker receives the latest gradient update from its predecessor and sends its gradients to its successor neighbor in a circular fashion. The trade-off between the synchronous and asynchronous implementation of SGD was exploited in [21]. Synchronous systems use the hardware resources less efficiently, whereas the asynchronous systems generally need more iterations since the gradient updates are computed on older versions of the model.

The use of specific hardware dedicated to deep learning seems, therefore, to be projected as a prosperous newborn branch for HPC. The introduction of the ultimate TPUs by Google research³ stands as an initial step in this direction. Notwithstanding, the range of hardware characteristics of multi-purpose supercomputers is very large. Scaling up computations might be cumbersome, requiring HPC expertise to tailor models on the available system hardware.

3 DATASETS AND METHODS

3.1 Datasets

We use the Camelyon 16 and 17 challenge data, which constitute, currently, one of the largest and most challenging datasets for histopathology research [4]. The datasets include WSIs of lymph node sections together with slide-level annotations of metastases type (negative, macro-metastases, micro-metastases, isolated tumor cells) and some manual segmentations of tumor regions. The data were collected at five different data centers, namely the University Medical Center in Nijmegen (RUMC), the Canisius-Wilhelmina Hospital in Nijmegen (CWZ), the University Medical Center Utrecht (UMCU), the Rijnstate Hospital in Arnhem (RST), and the Laboratory of Pathology East-Netherlands in Hengelo (LPON). A summary of the data provenances and distribution is given in Table 1.

The variability in preparation across acquisition centers makes the data very heterogeneous. Three different scanners were used in the five centers, namely the 3DHistech P250 (0.24 μm pixel size) at RUMC, CWZ and RST, the Philips IntelliSite Ultra Fast Scanner (0.25 μm pixel size) at LPON and the Hamamatsu XR C12000 (0.23 μm pixel size) at UMCU. The average file size is around 4 GB, for a total storage requirement of 3 030.5 GB.

³ <https://cloud.google.com/tpu/docs/tpus>

Year	Center	Total WSIs		Metastases (Train)			
		Train WSIs	Test WSIs	None	ITC	Micro	Macro
2016	RUMC	170	79	100	–	35	35
	UMCU	100	50	30	–	12	8
2017	CWZ	100	100	64	11	10	15
	RST	100	100	58	7	23	12
	UMCU	250	100	165	2	34	49
	RUMC	349	100	210	8	64	67
	LPON	100	100	61	8	5	26
Total		1 169	629	688	36	183	212

Table 1. WSI-level summary of the Camelyon 16 and 17 challenge datasets

3.2 Application-Driven Requirements

The design of the HPC infrastructure used to run the experiments was driven by the specific requirements of the pathology application, summarized in Table 2. The rapid growth of this field makes its requirements at the border of those of exascale computing both for computational and storage resources.

On the storage side, the extraction of image crops from the WSIs can easily grow to more than 60 thousand patches for a single record. With more than one record being held for each patient, the storage requirements can easily grow to several Terabytes (TB) of space. This highly demanding data preprocessing is shared with similar data types, e.g. satellite images. A requirement that is specific to the medical field, however, is the handling of sensible data. Large publicly available datasets such as Camelyon can be shared and downloaded on the main infrastructure to exploit the storage and computing facilities of the HPC providers. Sensible data such as private patient data must, however, be left on the local storage of the hospital to meet the security and privacy requirements. A typical approach, in this case, is that of the “Evaluation as a Service” (EaaS) solution, where the data can remain in a single infrastructure and does not need to be moved. Sensitive data could be left in the hospitals and be used only on their local environment.

On the computational side, the training of state-of-the-art CNNs with millions of parameters is highly demanding. Current implementations can take days to converge for datasets sizes of the order of magnitude of 10 Gigabytes (GB). Medical imaging data (and not only) easily reaches the TB if not the petabyte (PB) order of magnitude. The computational demand on such datasets can easily reach 15 petaflop/s [21], pointing towards exaflop/s in the nearest future. The support of dense linear algebra on distributed-memory HPC is an indispensable requirement to train deep models. Open Message Passing Interfaces and parallelization libraries such as Horovod⁴ allow the parallelization on multiple GPUs and HPC nodes. Top-development libraries such as Tensorflow and Keras are also top list

⁴ <https://eng.uber.com/horovod/>

requirements for the deployment of deep learning algorithms. Computational and storage requirements may seem two disentangled types of prerequisites, but one actually proportionally influences the other. With increasingly intense computations, the need for storage for saving intermediate results arises.

The need for containerized software is a further requirement to maintain the portability of the developments. While Docker containers are mostly used by the scientific community, Singularity containers constitute a better trade-off between the application and the infrastructure requirements, providing improved security by removing the root access on the HPC machines. Specific technologies are required to process the different image formats, being WSIs often saved as BIGTIFF files paired to annotation XMLs, CSVs or TXTs. Moreover, datasets such as PubMed central may require the handling of more common image compression formats such as JPEG and MPEG.

Requirement	Motivation	Software Layer
SCP and FTP connection	initial WSI transfer	1
Docker or Singularity	software portability	1, 2, 3
Data storage (> 100 TB)	public WSIs and intermediate results	1, 2
OpenSlides	WSI preprocessing	1
Tensorflow > 1.4.0	DL library	2, 3
Keras > 1.4.0	DL library	2, 3
Horovod	distributed computing	1, 2
OpenMPI	distributed computing	1, 2
SLURM	distributed computing	1

Table 2. Summary of application specific requirements

3.3 The Exascale Platform Architecture

In this section, we describe the architecture of the exascale platform within the developments for the PROCESS project, highlighting the modules introduced to adapt High-Performance Computing (HPC) to the use case requirements.

Data Services for Medical Use Case. The main data service of the PROCESS platform is a virtualized distributed file system (VDFS) LOBCDER. It is a modular, scalable and extensible VDFS implementing the micro-infrastructure approach. LOBCDER integrates special hardware nodes that are dedicated to the transfer of data. It has a module for meta-data management (DataNet) and pre/post-processing of exascale datasets (DISPEL).

Computing Services for the Medical Use Case. The PROCESS platform is available via Interactive Execution Environment (IEE) which gives access to heterogeneous computing infrastructure of supercomputers supporting HPC (via Rimrock) as well as cloud computing (via Cloudify). Both computing services provide relevant containerization services (Docker and Singularity). The

platform also allows users to use GPUs in parallel and in a distributed manner.

Table 3 summarizes the core requirements for the exascale platform given by the medical application. These requirements are one of the main pillars on which the PROCESS architecture was built. The table also provides an overview of how the PROCESS platform is capable to satisfy them.

Requirement	PROCESS Module
Support containerization	Rimrock
Workflow management for configuration, deployment and management of multiple application/use case executions	IEE
Distributed file system supporting medical use case datasets and their file formats (e.g. image formats)	LOBCDER
Support multiple pipelines within a workflow	LOBCDER, Rimrock
Distributed pre-processing of training data	DISPEL, LOBCDER
Supporting data transfer protocols (e.g. GridFTP, SCP, etc.)	LOBCDER
Support of the set of common tools for machine learning and deep learning on the HPC centres	Docker and Singularity containers supported by computing sites
Parallel and distributed multi-GPUs training	Computing centres
Support GPUs in containers	Computing centres

Table 3. Use case requirements and dedicated components from the PROCESS platform which fulfils them

3.4 A Modular Design for a Step-by-Step Pipeline

The method we propose is a three-layer software architecture for training different deep neural network models, summarized in Figure 2 and presented in detail in Figure 3. The modular approach splits the full pipeline in different workflows that can be shared, reused and updated independently. In the following subsections we describe the methods adopted in each of the three framework layers.

3.5 Layer 1: Preprocessing and Patch Extraction

The WSI in its original format is much larger than the maximum input format for a CNN. For this reason, WSIs need to be cropped into smaller images, called patches or tiles, which can then be fed to the network. Patches are extracted at the highest level of magnification (i.e. 40×). High resolution highlights qualitative features of the nuclei which are prognostic of cancer [30].

The first layer of the proposed application focuses on the patch extraction and data preprocessing of the WSIs (see Figure 4). As a first step, the filesystem is

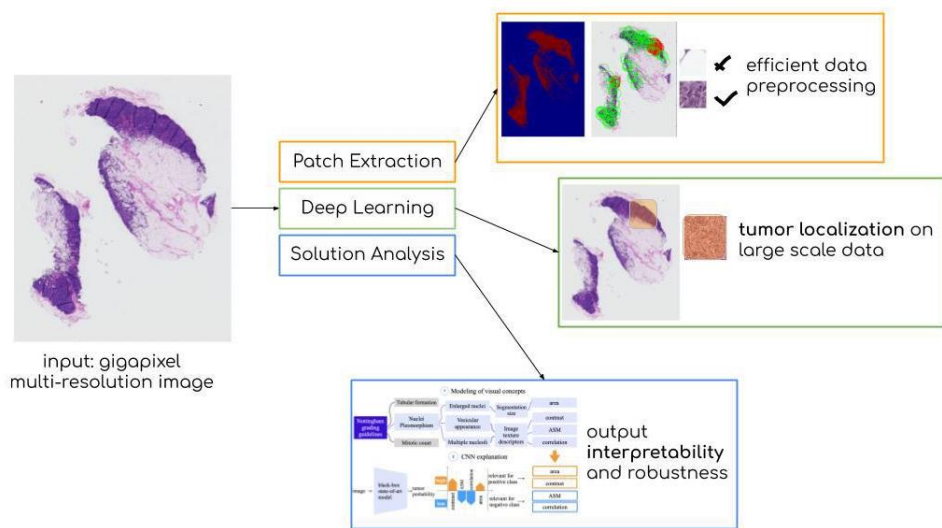


Figure 2. Overview of the CamNet software

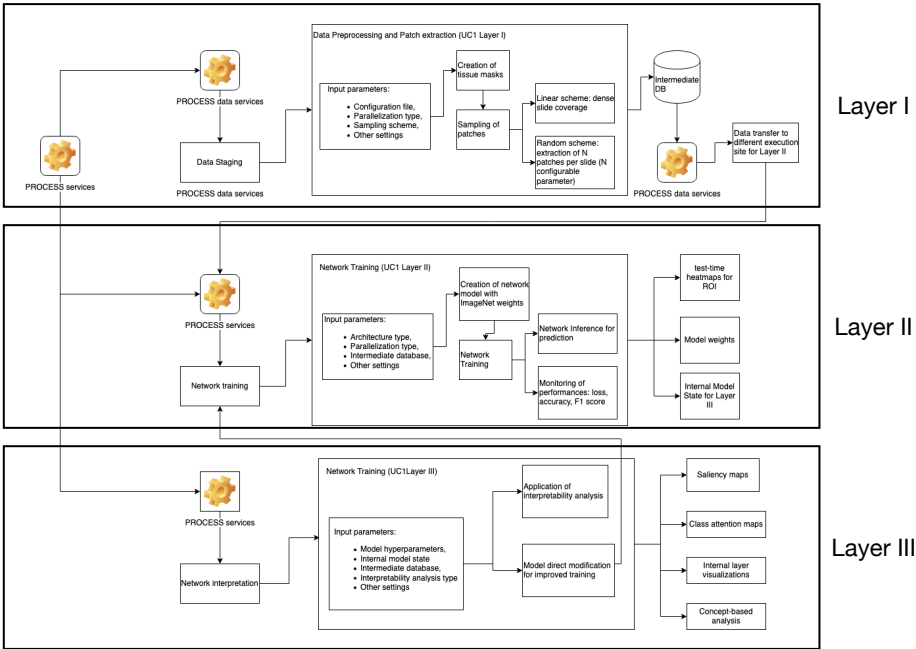


Figure 3. Detailed structure of each software layer. Best on screen.

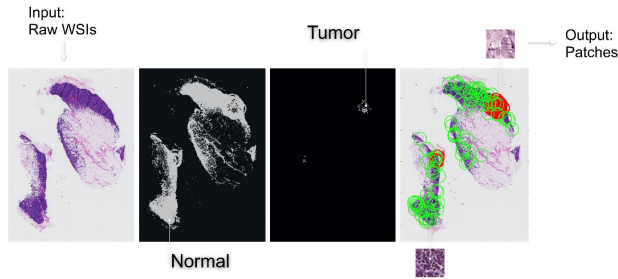


Figure 4. WSIs preprocessing pipeline: Normal tissue and tumor tissue masks are extracted and high-resolution patches are sampled from the selected regions

scanned to retrieve patient-related metadata and the acquisition center. If additional data were to be loaded on the HPC, the LOBCDER filesystem of PROCESS could be used. From the lowest magnification level, binary masks of normal and tumor tissue are extracted by the Otsu thresholding method [42], already proposed for this data in [4]. Patches are then sampled at the highest magnification level from the annotated tumor regions for the tumor class. For the non-tumor class, patches are sampled not only from the annotated images but also from 297 non-tumor WSIs in Camelyon17.

Patches with non-relevant information (e.g. white content, black pixels, background, etc.) are filtered out and discarded. Information about the patient, the lymph node, the hospital which handled the acquisitions, the resolution level of the patch and the patch location in the WSIs are stored together with the pixel values in an intermediate HDF5 database. Moreover, the doctor annotations are stored in the HDF5 as a binary label on the patch, which discriminates between tumor and non-tumor patches. Different cropping strategies following the sampling approaches for automated histological image analysis [4] are available in the system. The most common is the random sampling scheme, which extracts randomly several patch locations. A seed for initializing the random sampling and the desired number of patches to extract (N) are passed as input parameters of this stage. A white-threshold (expressed in terms of the percentage of pixels with intensity values larger than 200) is applied to discard image croppings of the background and the adipose tissue, which are uninformative for the task. Similarly, black background patches are removed from the sampling results. The Simple Linux Utility for Resource Management system (SLURM) is used to develop a parallel version that can be distributed on the HPC. A job array is configured with a random generator seed. At each batch run, a different patch set is extracted. The code for local execution (sequential algorithm) and for the distributed execution is available online⁵.

⁵ https://github.com/medgift/PROCESS_L1

By means of the HPC computational capacity, the dense coverage of the WSIs is also possible. A sliding window that extracts patches with a fixed stride is implemented as an alternative sampling option. This option, not possible with the local research facilities, is optimal when ran on the computing capabilities of HPC. The distribution of the code on different HPC nodes is, in this case, necessary. Therefore, this part of the software can only be run on distributed computing facilities with the support of SLURM. Each WSI is assigned to a single SLURM job array. The patches with non-relevant information are filtered out and discarded by the white thresholding. A further scaling step is also available, combining two SLURM techniques at the same time for better efficiency. Each task in the SLURM job array is assigned a different WSI, and an arbitrary number of subtasks are run in parallel to implement the sliding window.

The image croppings resulting from the extraction process with any of the two strategies are stored in an intermediate Hierarchical Data Format file (HDF5), together with the metadata about the patient, the lymph node, the acquisition center, the magnification level and the patch location.

3.6 Layer 2: Patch-Based Deep Learning

The second layer loads the intermediate HDF5 dataset generated by Layer 1, and focuses on the training of deep learning architectures for the binary classification between tumor and non-tumor patches, following the approach in [42]. The training of several state-of-the-art CNNs (i.e. ResNet50, ResNet101, GoogleNet, Inception V3, DenseNet, InceptionResNetV2, all pre-trained on ImageNet) is pre-implemented. The training can be distributed with the Open Source Distributed Deep Learning Framework for Tensorflow, Horovod. A configuration file is used to specify the network architecture preferences and hyperparameters (i.e. loss, activation, learning rate and batch size). The output of this layer consists of the trained network parameters and training statistics. In this paper, we compare the performances of ResNet and Inception on different GPU types, namely Titan V, Titan X, Tesla K80, Tesla K40. The source code is also available online, for either single-GPU or multi-GPU execution⁶.

3.7 Layer 3: Inference and Interpretability

We present a summary of the functionalities of Layer III in Figure 5. This last layer of the proposed application deals with two main tasks: performing inference and interpreting the models trained in Layer II.

As a first functionality, this layer generates heatmaps of the probability of the presence of tumorous tissue, which can be used for visual inspection and comparison with the ground truth segmentations. CNN inference, in general, is less costly than training, although WSIs still require a great number of patches to be tested. This

⁶ https://github.com/medgift/PROCESS_L2/

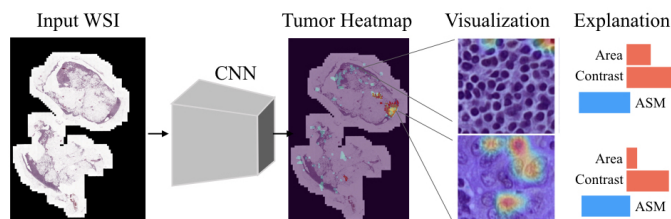


Figure 5. An example of the functionalities in Layer III, which consist of performing distributed inference for building the tumor heatmap and providing interpretability analyses and insights about network training

process is hence optimized by parallel-data distribution. Several copies of the CNN are stored on the GPUs and inference is performed in parallel on distinct batches of data. The heatmaps are then built by interpolating the predicted probability values for each pixel.

The interpretability of the models, besides, is analyzed in this layer. Understanding the decision-making process of CNNs is a key point in medical imaging, to ensure that clinically correct decisions are taken. Several different approaches are proposed in the literature, with clear distinctions being made between models that introduce interpretability as a built-in additional task [11, 20, 6, 8, 2, 34] and post-hoc methods. Post-hoc methods, as defined in [24], are particularly suited to the proposed layered framework, since they allow to disentangle the interpretability analysis from network training. They can be used to explain any machine learning algorithm without retraining the network weights. Several post-hoc techniques [36, 43, 33, 10, 31] highlight the most influential set of features in the input space, a technique known as attribution to features [37]. Among these, gradient-based approaches, such as Class Activation Maps (CAM), Gradient-weighted Class Activation Maps (grad-CAM) and its improved version grad-CAM++, attribute the network output to perturbations of the input pixels. The output of such methods is a heatmap of the input pixels that mostly contributed to the decision. Local Interpretable Model-Agnostic Explanations (LIME) are used as an alternative tool to obtain visualizations. These visualizations are compared to interpreting the CNNs by the regression of clinical values such as lesion extension and appearance in the internal network activations in [15]. The method proposed for the comparison is that of Regression Concept Vectors (RCVs), first implemented in [14] and then expanded in [17]. This approach, besides, was further investigated in [16], showing that it can be efficiently used to improve the training of CNNs on medical images. To develop concept-based explanations, we define a list of concepts that could be relevant in the diagnosis. Concepts are chosen so that specific questions can be addressed, e.g.: *Do the nuclei appear larger than usual?* *Do they present a vesicular texture with high chromatism?* To answer these questions about the nuclei area and texture, representative of the NGH nuclear pleomorphism, the segmentation

of the nuclei instances in the image is obtained by either manual annotations [14] or automatic segmentation [27]. We express the nuclei area as the sum of pixels in the nuclei contours, whereas the nuclei texture is described by Haralick’s descriptors such as Angular Second Moment (ASM), contrast and correlation [18]. A summary of the concepts extracted and how to compute them is presented in Figure 6.

The performance of the RCV is evaluated by the determination coefficient of the regression R^2 , expressing the percentage of variation that is captured by the regression. This is used to check if the network is learning the concepts and in which network layers [14, 17]. Sensitivity scores are computed on testing patches from Camelyon17 as in [14]. The global relevance of the concept is estimated by either TCAV or Br scores.






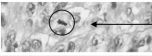


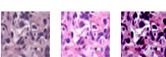
Concept	Clinical reference	Description	Visual examples		Magnification	Source	Type
Count of cavities	NGH tubular formation	tumour cells in gland structure	 well formed	 poorly formed	low	annotation or automated	D
Nuclei area	NGH nuclear pleomorphism	abnormality in size	 regular	 enlarged	high	annotation or automated	C
Nuclei Texture		vesicular appearance		 uneven stain			
Mitotic count	NGH mitotic count	number of mitosis	 mitosis		high	annotation or automated	D
Nuclei density	Ki-67 protein expression	cell proliferation	 regular	 overgrowth	any	annotation or automated	D
Staining	Staining procedure	dye applied on the tissues	 different appearance		any	metadata	D

Figure 6. Concept list derived for the breast histopathology application with information about the magnification level at which to extract them and whether the measures require a continuous (C) or discrete (D) representation

4 EVALUATION

4.1 Data Preprocessing

We report in Table 4 the evaluation of the execution times for the data preprocessing and patch extraction workflow. The two sampling processes, random and dense, are compared. The measurements were computed on the AGH site in Krakow, Poland.

The layer scalability to increasingly larger datasets and patient cohorts are shown in Figure 7. Scaling is possible by increasing the number of available nodes, for instance, from 100 to 1000. In this case, approximately 50 000 patches can be extracted in less than 5 minutes with random sampling strategy, showing lin-

# WSIs	# Nodes	Sampling	Parallelization Type	CPU Time [s]
1	1	random	none	292
5	5	random	1 CPU/WSI	260
1	1	dense	none	18 400
1	100	dense	1 000 CPUs/WSI	3 000
5	500	dense	5 000 CPUs/WSI	7 530
5	1 000	dense	10 000 CPUs/WSI	3 780

Table 4. Measurements of execution time vs data sizes for extracting high resolution patches from the Camleyon17 dataset at the PROCESS AGH site. The WSI size is $100\,000 \times 100\,000$ pixels.

ear speed-up capability. Through the dense sampling strategy and by scaling the patch extraction to 1000 nodes, we extracted 43 million patches, for a total of 7TB of intermediate data. With 8 thousand nodes available for the computation, this would take approximately 1 hour with the SLURM parallelization technique.

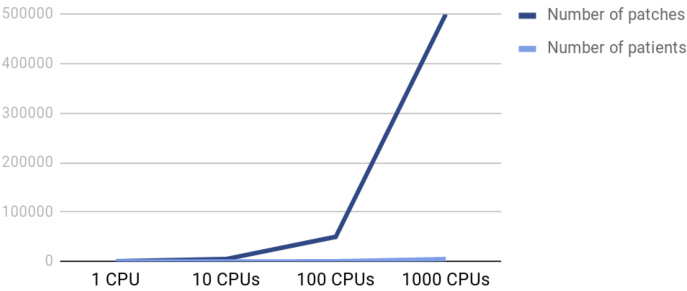


Figure 7. Layer 1 scalability to larger datasets and patient cohorts vs number of available nodes

4.2 Data Transfer Between HPC Sites

In addition to the computational time, we evaluate the time for data transfer between HPC centers, to establish whether this could constitute a possible bottleneck that would prevent the execution of two different software layers in two centers, e.g. Layer I at AGH and Layer II at UVA.

We show the cross-site staging results for transferring 30 GB of the Camleyon 16 dataset (3 % of the full dataset size, approximatively) in Table 5. Where available, i.e. for LRZ and UVA, we compare the Data Transfer Nodes connections. DTN nodes speed up trasfers of nearly 30 % compared to the standard SCP protocol.

	LRZ DTN	UVA DTN	LISA	AGH
LRZ DTN	–	405.32	25.53	32.17
UVA DTN	494.51	–	324.17	48.60
LISA	324.97	549.62	–	30.27
AGH	14.71	51.07	30.27	–

Table 5. Cross-site data staging speed for transferring 3 % of the Camelyon data with the gridFTP protocol. Measures are reported in Mb/s.

Figure 8 compares using standard SCP protocols for data transfer between DTN nodes against the dynamic reprogramming of DTNs with the FDT protocol through containers. For files smaller than 2 Gb, the overhead of deploying the containers on the fly is greater than the transfer time. For larger files, however, the overhead is amortized by the better performing FDT protocol.

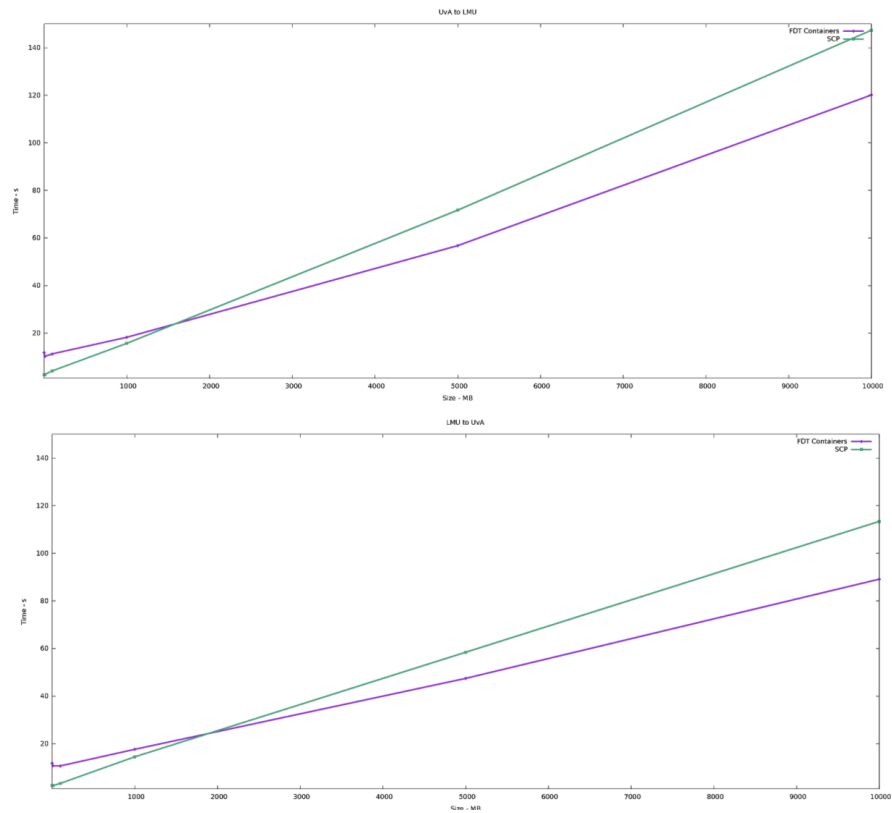


Figure 8. Comparison of SCP protocols vs FTD containerized approach for data transfer

4.3 Model Training

We compare in Figure 9 the training times (over 10 epochs) of two different architectures, namely ResNet50 and InceptionV3, on 50 Gb of training data. Less performant GPUs require a longer time to perform the training operations, with NVIDIA K80 requiring more than 7 hours to train the 26 million of parameters of ResNet50. This time reduces to slightly more than 1 hour when using the latest NVIDIA V100. The model parallelization on two GPUs, particularly on 2 NVIDIA V100 shows the scalability of the network training over multiple GPUs, requiring 1 hour and a half to train the 24 million of parameters of Inception V3.

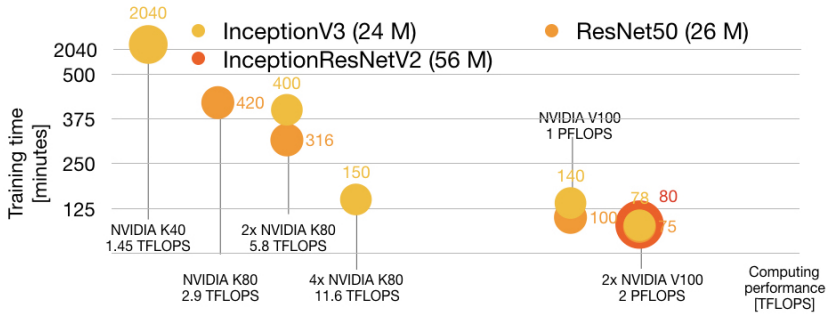


Figure 9. Comparison of ResNet and Inception average training times (on 50 Gb of training data) with single and distributed training on different GPUs. The number of parameters being trained is reported in brackets (M = millions). The floating point operations per second (FLOPS) are reported for each GPU configuration. The size of the circle is proportional to the number of parameters in each network.

The scalability of network training over larger datasets is compared for ResNet50 distributed on 2 NVIDIA V100 in Figure 10. This is insightful about the scalability of the combination of Layer I and Layer II estimating the training time per epoch for increasing dataset sizes.

4.4 Visualizations and Interpretability

Figure 11 shows some output heatmaps overlayed to the original input WSIs and compared to the manual tumor segmentations provided by the pathologist. The inference of nearly 10 thousand patches is distributed over 5 processes on a single NVIDIA V100, requiring less than 4 minutes to compute the heatmap for an input WSI (230s). The network output is interpreted at the patch-level using gradCAM, gradCAM++ and LIME. Some examples are shown in Figure 11.

The concepts representative of the NGH nuclear pleomorphism are learned in early layers of the network, as shown in [14, 17]. Particularly, from the analysis of

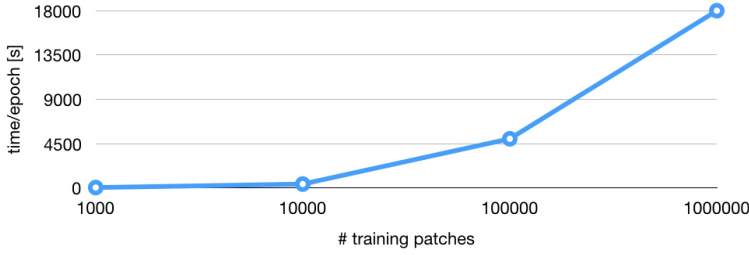


Figure 10. Training time per epoch vs increasingly larger data sizes for ResNet50 on $2 \times$ NVIDIA V100

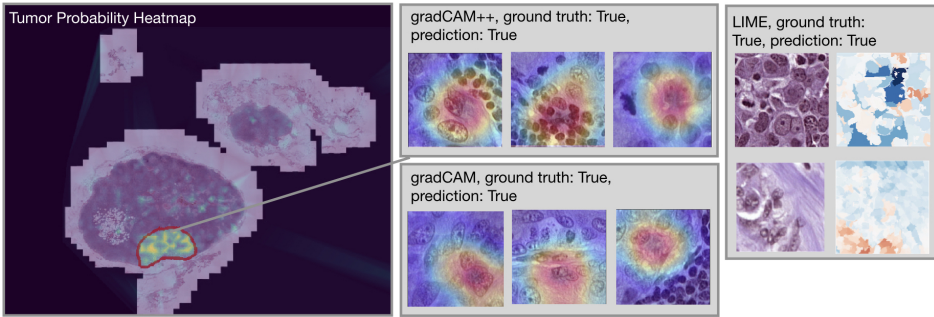


Figure 11. Visualization outputs. The heatmaps of tumor probability are computed by distributed inference over 5 models replicas on a single NVIDIA V100. The interpretability visualizations (on the right) were also computed on the same GPU.

the TCAV and Br scores it emerges that the contrast of the nuclei texture is relevant to the classification, with TCAV = 0.72 out of 1 and Br = 0.27.

5 DISCUSSION

The best performance for the data preprocessing and patch extraction pipeline (layer 1) is obtained when this layer is run on the HPC site. The scaling up of the dataset sizes is possible under the condition of a sufficiently large number of CPU cores on the computational site, which narrows down the computational time required by each operation. The results in Table 4 show the large benefits of optimizing the data extraction process on the HPC resources. In less than one hour, nearly 1 TB of data were extracted from 5 WSIs, with a computational requirement of one thousand nodes (with 10 CPU cores per node). Scaling this up led us to the extraction of 34 million of patches for a total of 7 TB that can be used for network training. Under the hypothesis of a large number of CPUs available, i.e. one per WSI, the computational requirements will not be a limit for data preprocessing, as the scalability requirements are almost linear.

Similarly, layer 2 was tested on different GPU servers, comparing performances of state-of-the-art networks and GPU types. The containerized approach allows users to deploy each layer on multiple sites without requiring specific expertise on the deployment site, thus leaving open different possibilities for deployment. The training times are consistently narrowed down by the distribution of training with model parallelism on multiple GPUs, as shown in Figures 9 and 10. The results on the distribution of network training show that the TFLOP of the GPUs is not a major limitation for scaling to larger datasets, with the performance of 4 NVIDIA K80 being close to that of a single NVIDIA V100 (150 minutes against 140 minutes respectively, as shown in Figure 9). Increasing data sizes leads to longer training times per epoch, as expected. In this case, even more TFLOPS are needed to scale up to millions of images. The training on 1 million images, however, can be performed in approximately 2 days with parallelization on 2 NVIDIA V100. The heatmap generation and interpretability analyses provide a direct visualization of the regions with a high probability of being tumorous. GradCAM, gradCAM++ and LIME provide various insights about the input regions responsible for the decision. RCVs further showed the importance of nuclei texture in the classification, with nuclei texture contrast being particularly relevant to the classification of patches of breast tissue. This is in accordance with the NHG grading system, which identifies hyperchromatism as a signal of nuclear atypia. It seems therefore that nuclear pleomorphism is taken into consideration during network training.

The results for each layer suggest that an optimal configuration would make the best use of the CPU clusters for data preprocessing, while network training should be performed on GPU servers. By testing data transfer times (see Table 5) we showed that the FTD containerized approach with DTNs would reduce the data staging bottleneck to the minimum.

6 CONCLUSION

We proposed a modular application that adapts with large flexibility to the different requirements of research in deep learning for histopathology and is deployable on HPC computing. The three layers can be deployed independently on the appropriate computing site to run different parts of the pipeline. The modularity of the proposed application embraces the foreseen future of digital pathology, being easy to deploy and offering large customization in terms of network parameters and choice of the training data. The parallelization of the different workflows improves the performance in terms of computational time. This is in line with the requirements of digital pathology, that, with increasingly larger datasets being collected and thus increasingly demanding tasks being set, is becoming demanding in terms of computational and storage requirements.

Moreover, the network training could be deployed independently on a private computational site, allowing users to fine-tune the network weights on sensitive data that cannot be shared.

Acknowledgements

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”.

REFERENCES

- [1] AEFFNER, F.—ZARELLA, M. D.—BUCHBINDER, N.—BUI, M. M.—GOODMAN, M. R.—HARTMAN, D. J.—LUJAN, G. M.—MOLANI, M. A.—PARWANI, A. V.—LILLARD, K.—TURNER, O. C.—VEMURI, V. N. P.—YUIL-VALDES, A. G.—BOWMAN, D.: Introduction to Digital Image Analysis in Whole-Slide Imaging: A White Paper from the Digital Pathology Association. *Journal of Pathology Informatics*, Vol. 10, 2019, No. 9, doi: 10.4103/jpi.jpi.82.18.
- [2] ALVAREZ-MELIS, D.—JAAKKOLA, T. S.: Towards Robust Interpretability with Self-Explaining Neural Networks. *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS 2018)*, 2018, pp. 7786–7795.
- [3] BONERT, M.—TATE, A. J.: Mitotic Counts in Breast Cancer Should Be Standardized with a Uniform Sample Area. *BioMedical Engineering OnLine*, Vol. 16, 2017, No. 1, Art. No. 28, doi: 10.1186/s12938-016-0301-z.
- [4] BÁNDI, P.—GEESINK, O.—MANSON, Q.—VAN DIJK, M.—BALKENHOL, M.—HERMSEN, M.—EHTESHAMI BEJNORDI, B.—LEE, B.—PAENG, K.—ZHONG, A.—LI, Q.—ZANJANI, F. G.—ZINGER, S.—FUKUTA, K.—KOMURA, D.—OVTCHAROV, V.—CHENG, S.—ZENG, S.—THAGAARD, J.—DAHL, A. B.—LIN, H.—CHEN, H.—JACOBSSON, L.—HEDLUND, M.—ÇETIN, M.—HALICI, E.—JACKSON, H.—CHEN, R.—BOTH, F.—FRANKE, J.—KÜSTERS-VANDEVELDE, H.—VREULS, W.—BULT, P.—VAN GINNEKEN, B.—VAN DER LAAK, J.—LITJENS, G.: From Detection of Individual Metastases to Classification of Lymph Node Status at the Patient Level: The CAMELYON17 Challenge. *IEEE Transactions on Medical Imaging*, Vol. 38, 2019, No. 2, pp. 550–560, doi: 10.1109/TMI.2018.2867350.
- [5] CAMPBELL, C.—MECCA, N.—DUONG, T.—OBEID, I.—PICONE, J.: Expanding an HPC Cluster to Support the Computational Demands of Digital Pathology. *2018 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, 2018, doi: 10.1109/SPMB.2018.8615614.
- [6] CARUANA, R.—LOU, Y.—GEHRKE, J.—KOCH, P.—STURM, M.—ELHADAD, N.: Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-Day Readmission. *Proceedings of the 21th ACM SIGKDD International Conference on*

- Knowledge Discovery and Data Mining (KDD '15), ACM, 2015, pp. 1721–1730, doi: 10.1145/2783258.2788613.
- [7] CHAGPAR, A.—MIDDLETON, L. P.—SAHIN, A. A.—MERIC-BERNSTAM, F.—KUERER, H. M.—FEIG, B. W.—ROSS, M. I.—AMES, F. C.—SINGLE-TARY, S. E.—BUCHHOLZ, T. A.—VALERO, V.—HUNT, K. K.: Clinical Outcome of Patients with Lymph Node-Negative Breast Carcinoma Who Have Sentinel Lymph Node Micrometastases Detected by Immunohistochemistry. *Cancer*, Vol. 103, 2005, No. 8, pp. 1581–1586, doi: 10.1002/cncr.20934.
 - [8] CHO, K.—COURVILLE, A.—BENGIO, Y.: Describing Multimedia Content Using Attention-Based Encoder-Decoder Networks. *IEEE Transactions on Multimedia*, Vol. 17, 2015, No. 11, pp. 1875–1886, doi: 10.1109/TMM.2015.2477044.
 - [9] DEAN, J.—CORRADO, G.—MONGA, R.—CHEN, K.—DEVIN, M.—MAO, M.—RANZATO, M.—SENIOR, A.—TUCKER, P.—YANG, K.—LE, Q.—NG, A.: Large Scale Distributed Deep Networks. In: Pereira, F., Burges, C. J. C., Bottou, L. Weinberger, K. Q. (Eds.): *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, 2012, pp. 1223–1231.
 - [10] FONG, R. C.—VEDALDI, A.: Interpretable Explanations of Black Boxes by Meaningful Perturbation. *Proceedings of the 2017 IEEE International Conference on Computer Vision, Venice, Italy, 2017*, pp. 3449–3457, doi: 10.1109/ICCV.2017.371.
 - [11] FREITAS, A. A.: Comprehensible Classification Models: A Position Paper. *ACM SIGKDD Explorations Newsletter*, Vol. 15, 2014, No. 1, pp. 1–10, doi: 10.1145/2594473.2594475.
 - [12] GIULIANO, A. E.—BALLMAN, K. V.—MCCALL, L.—BEITSCH, P. D.—BRENNAN, M. B.—KELEMEN, P. R.—OLLILA, D. W.—HANSEN, N. M.—WHITWORTH, P. W.—BLUMENCRAZ, P. W.—LEITCH, A. M.—SAHA, S.—HUNT, K. K.—MORROW, M.: Effect of Axillary Dissection vs. No Axillary Dissection on 10-Year Overall Survival Among Women with Invasive Breast Cancer and Sentinel Node Metastasis: The ACOSOG Z0011 (Alliance) Randomized Clinical Trial. *JAMA*, Vol. 318, 2017, No. 10, pp. 918–926, doi: 10.1001/jama.2017.11470.
 - [13] GOYAL, P.—DOLLÁR, P.—GIRSHICK, R.—NOORDHUIS, P.—WESOŁOWSKI, L.—KYROLA, A.—TULLOCH, A.—JIA, Y.—HE, K.: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. 2017, arXiv preprint arXiv:1706.02677.
 - [14] GRAZIANI, M.—ANDREARCYK, V.—MÜLLER, H.: Regression Concept Vectors for Bidirectional Explanations in Histopathology. In: Stoyanov, D. et al. (Eds.): *Understanding and Interpreting Machine Learning in Medical Image Computing Applications (MLCN 2018, DLF 2018, IMIMIC 2018)*. Springer, Cham, Lecture Notes in Computer Science, Vol. 11038, 2018, pp. 124–132, doi: 10.1007/978-3-030-02628-8.14.
 - [15] GRAZIANI, M.—ANDREARCYK, V.—MÜLLER, H.: Visualizing and Interpreting Feature Reuse of Pretained CNNs for Histopathology. *Irish Machine Vision and Image Processing Conference (IMVIP 2019)*, Dublin, Ireland, 2019.
 - [16] GRAZIANI, M.—LOMPECH, T.—MÜLLER, H.—DEPEURSINGE, A.—ANDREARCYK, V.: Interpretable CNN Pruning for Preserving Scale-Covariant Features in Medical Imaging. In: Cardoso, J. et al. (Eds.): *Interpretable and Annotation-Efficient Learning for Medical Image Computing (IMIMIC 2020, MIL3ID*

- 2020, LABELS 2020). Springer, Cham, in cooperation with MICCAI, Lecture Notes in Computer Science, Vol. 12446, 2020, pp. 23–32, doi: 10.1007/978-3-030-61166-8_3.
- [17] GRAZIANI, M.—ANDREARCZYK, V.—MARCHAND-MAILLET, S.—MÜLLER, H.: Concept Attribution: Explaining CNN Decisions to Physicians. *Computers in Biology and Medicine*, Vol. 123, 2020, Art.No. 103865, doi: 10.1016/j.combiomed.2020.103865.
- [18] HARALICK, R. M.—SHANMUGAM, K.—DINSTEIN, I.: Textural Features for Image Classification. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-3, 1973, No. 6, pp. 610–621, doi: 10.1109/TSMC.1973.4309314.
- [19] HAYAKAWA, T.—PRASATH, V. B. S.—KAWANAKA, H.—ARONOW, B. J.—TSURUOKA, S.: Computational Nuclei Segmentation Methods in Digital Pathology: A Survey. *Archives of Computational Methods in Engineering*, 2019, pp. 1–13, doi: 10.1007/s11831-019-09366-4.
- [20] KIM, B.—SHAH, J. A.—DOSHI-VELEZ, F.: Mind the Gap: A Generative Approach to Interpretable Feature Selection and Extraction. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (Eds.): *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, 2015, pp. 2260–2268.
- [21] KURTH, T.—ZHANG, J.—SATISH, N.—RACAH, E.—MITLIAGKAS, I.—PATWARY, M. M. A.—MALAS, T.—SUNDARAM, N.—BHIMJI, W.—SMORKALOV, M.—DESLIPPE, J.—SHIRYAEV, M.—SRIDHARAN, S.—PRABHAT—DUBEY, P.: Deep Learning at 15PF: Supervised and Semi-Supervised Classification for Scientific Data. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC’17)*, ACM, 2017, Art.No. 7, 11 pp., doi: 10.1145/3126908.3126916.
- [22] LI, J.—YANG, S.—HUANG, X.—DA, Q.—YANG, X.—HU, Z.—DUAN, Q.—WANG, C.—LI, H.: Signet Ring Cell Detection with a Semi-Supervised Learning Framework. In: Chung, A., Gee, J., Yushkevich, P., Bao, S. (Eds.): *Information Processing in Medical Imaging (IPMI 2019)*. Springer, Cham, Lecture Notes in Computer Science, Vol. 11492, 2019, pp. 842–854, doi: 10.1007/978-3-030-20351-1_66.
- [23] LIN, Y.—HAN, S.—MAO, H.—WANG, Y.—DALLY, W. J.: Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. 2017, arXiv preprint arXiv:1712.01887.
- [24] LIPTON, Z. C.: The Mythos of Model Interpretability. *Communication of ACM*, Vol. 61, 2018, No. 10, pp. 36–43, doi: 10.1145/3233231.
- [25] LU, C.—ROMO-BUCHELI, D.—WANG, X.—JANOWCZYK, A.—GANESAN, S.—GILMORE, H.—RIMM, D.—MADABHUSHI, A.: Nuclear Shape and Orientation Features from H & E Images Predict Survival in Early-Stage Estrogen Receptor-Positive Breast Cancers. *Laboratory Investigation*, Vol. 98, 2018, No. 11, pp. 1438–1448, doi: 10.1038/s41374-018-0095-7.
- [26] MADABHUSHI, A.—LEE, G.: Image Analysis and Machine Learning in Digital Pathology: Challenges and Opportunities. *Medical Image Analysis*, Vol. 33, 2016, pp. 170–175, doi: 10.1016/j.media.2016.06.037.

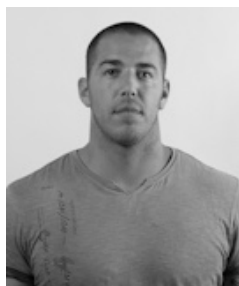
- [27] OTÁLORA, S.—ATZORI, M.—KHAN, A.—JIMENEZ-DEL-TORO, O.—ANDREARCYK, V.—MÜLLER, H.: Systematic Comparison of Deep Learning Strategies for Weakly Supervised Gleason Grading. *Medical Imaging 2020: Digital Pathology. Proceedings of the SPIE*, Vol. 11320, 2020, Art.No. 113200L, doi: 10.1117/12.2548571.
- [28] OTÁLORA, S.—MARINI, N.—MÜLLER, H.—ATZORI, M.: Semi-Weakly Supervised Learning for Prostate Cancer Image Classification with Teacher-Student Deep Convolutional Networks. In: Cardoso, J. et al. (Eds.): *Interpretable and Annotation-Efficient Learning for Medical Image Computing (IMIMIC 2020, MIL3ID 2020, LABELS 2020)*. Springer, Cham, *Lecture Notes in Computer Science*, Vol. 12446, 2020, pp. 193–203, doi: 10.1007/978-3-030-61166-8_21.
- [29] RABE, K.—SNIR, O. L.—BOSSUYT, V.—HARIGOPAL, M.—CELLI, R.—REISENBICHLER, E. S.: Interobserver Variability in Breast Carcinoma Grading Results in Prognostic Stage Differences. *Human Pathology*, Vol. 94, 2019, pp. 51–57, doi: 10.1016/j.humpath.2019.09.006.
- [30] RAKHA, E. A.—EL-SAYED, M. E.—LEE, A. H. S.—ELSTON, C. W.—GRAINGE, M. J.—HODI, Z.—BLAMEY, R. W.—ELLIS, I. O.: Prognostic Significance of Nottingham Histologic Grade in Invasive Breast Carcinoma. *Journal of Clinical Oncology*, Vol. 26, 2008, No. 19, pp. 3153–3158, doi: 10.1200/JCO.2007.15.5986.
- [31] RIBEIRO, M. T.—SINGH, S.—GUESTRIN, C.: Why Should I Trust You?: Explaining the Predictions of Any Classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, ACM, 2016, pp. 1135–1144, doi: 10.1145/2939672.2939778.
- [32] SCHNITT, S. J.—CONNOLLY, J. L.—TAVASSOLI, F. A.—FECHNER, R. E.—KEMPSON, R. L.—GELMAN, R.—PAGE, D. L.: Interobserver Reproducibility in the Diagnosis of Ductal Proliferative Breast Lesions Using Standardized Criteria. *The American Journal of Surgical Pathology*, Vol. 16, 1992, No. 12, pp. 1133–1143, doi: 10.1097/00000478-199212000-00001.
- [33] SELVARAJU, R. R.—COGSWELL, M.—DAS, A.—VEDANTAM, R.—PARIKH, D.—BATRA, D.: Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 618–626, doi: 10.1109/ICCV.2017.74.
- [34] SHEN, S.—HAN, S. X.—ABERLE, D. R.—BUI, A. A.—HSU, W.: An Interpretable Deep Hierarchical Semantic Convolutional Neural Network for Lung Nodule Malignancy Classification. *Expert Systems with Applications*, Vol. 128, 2019, pp. 84–95, doi: 10.1016/j.eswa.2019.01.048.
- [35] SIEGEL, R. L.—MILLER, K. D.—JEMAL, A.: *Cancer Statistics, 2019*. CA: A Cancer Journal for Clinicians, Vol. 69, 2019, No. 1, pp. 7–34, doi: 10.3322/caac.21551.
- [36] SIMONYAN, K.—VEDALDI, A.—ZISSERMAN, A.: Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *Computing Research Repository (CoRR)*, 2013, arXiv:1312.6034, <http://arxiv.org/abs/1312.6034>.

- [37] SUNDARARAJAN, M.—TALY, A.—YAN, Q.: Axiomatic Attribution for Deep Networks. Proceedings of the 34th International Conference on Machine Learning (ICML '17), JMLR.org, Proceedings of Machine Learning Research (PMLR), Vol. 70, 2017, pp. 3319–3328.
- [38] VAN DIEST, P. J.—VAN DEURZEN, C. H. M.—CSERNI, G.: Pathology Issues Related to SN Procedures and Increased Detection of Micrometastases and Isolated Tumor Cells. *Breast Disease*, Vol. 31, 2010, No. 2, pp. 65–81, doi: 10.3233/BD-2010-0298.
- [39] WANG, F.—AJI, A.—LIU, Q.—SALTZ, J. H.: Hadoop-GIS: A High Performance Query System for Analytical Medical Imaging with Mapreduce. Technical Report CCI-TR-2001-3, Emory University, Atlanta, USA, 2011, pp. 1–13.
- [40] WEI, X. S.—WU, J.—ZHOU, Z. H.: Scalable Multi-Instance Learning. 2014 IEEE International Conference on Data Mining (ICDM), Shenzhen, China, 2014, pp. 1037–1042, doi: 10.1109/ICDM.2014.16.
- [41] XU, Y.—LI, Y.—SHEN, Z.—WU, Z.—GAO, T.—FAN, Y.—LAI, M.—CHANG, E. I.-C.: Parallel Multiple Instance Learning for Extremely Large Histopathology Image Analysis. *BMC Bioinformatics*, Vol. 18, 2017, Art.No. 360, 15 pp., doi: 10.1186/s12859-017-1768-8.
- [42] ZANJANI, F. G.—ZINGER, S.—DE, P. N.: Automated Detection and Classification of Cancer Metastases in Whole-Slide Histopathology Images Using Deep Learning. 2017.
- [43] ZEILER, M. D.—FERGUS, R.: Visualizing and Understanding Convolutional Networks. Computing Research Repository (CoRR), 2013, arXiv:1311.2901, <http://arxiv.org/abs/1311.2901>.



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.



Ivan EGGEL is Senior Research Associate of the University of Applied Sciences of Western Switzerland. His research interest focuses on developing applications for information retrieval in healthcare and in cloud computing. He participated in the organization of several challenges such as ImageCLEF and VIS-CERAL.



François DELIGAND is a Bachelor's student in mathematics and informatics at INP-ENSEEIH in Toulouse, France. He contributed to the experiments during his internship at HES-SO Valais.



Vincent ANDREARCZYK is currently Senior Researcher at the University of Applied Sciences and Arts Western Switzerland with a research focus on deep learning for medical image analysis and texture feature extraction. He received a double Masters degree in electronics and signal processing from ENSEEIHT, France and Dublin City University in 2012 and 2013, respectively. He completed his Ph.D. degree on deep learning for texture and dynamic texture analysis at Dublin City University in 2017.



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



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 coordinator of the Khresmoi EU project, scientific coordinator of the VISCERAL EU project and is the initiator of the ImageCLEF benchmark that has run medical tasks since 2004. He has authored over 600 scientific papers with more than 17 000 citations and is in the editorial board of several journals.

ALLSCALE API

Philipp GSCHWANDTNER, Herbert JORDAN
Peter THOMAN, Thomas FAHRINGER

*Department of Computer Science, University of Innsbruck
Technikerstrasse 21a, 6020 Innsbruck, Austria*

e-mail: {philipp.gschwandtner, herbert.jordan, peter.thoman,
thomas.fahringer}@uibk.ac.at

Abstract. Effectively implementing scientific algorithms in distributed memory parallel applications is a difficult task for domain scientists, as evident by the large number of domain-specific languages and libraries available today attempting to facilitate the process. However, they usually provide a closed set of parallel patterns and are not open for extension without vast modifications to the underlying system. In this work, we present the AllScale API, a programming interface for developing distributed memory parallel applications with the ease of shared memory programming models. The AllScale API is closed for a modification but open for an extension, allowing new user-defined parallel patterns and data structures to be implemented based on existing core primitives and therefore fully supported in the AllScale framework. Focusing on high-level functionality directly offered to application developers, we present the design advantages of such an API design, detail some of its specifications and evaluate it using three real-world use cases. Our results show that AllScale decreases the complexity of implementing scientific applications for distributed memory while attaining comparable or higher performance compared to MPI reference implementations.

Keywords: API, programming interface, parallel programming, shared memory, distributed memory, parallel operator, data structure

Mathematics Subject Classification 2010: 68-W10

1 INTRODUCTION

Even with the recent trend of many-core processors providing users with dozens of cores per chip in a single memory address space, distributed memory systems pose an essential aspect of HPC in order to achieve large-scale performance for scientific applications. Although there are certain system architectures that overcome the issue of distinct memory address spaces by hardware means (e.g. SGI's UV [21] series using the NumaLink protocol), the conventional approach is still to handle distinct memory address spaces in the software stack by providing a global address space in software or by explicit message exchange.

However, most of these ubiquitous software solutions entail several disadvantages that make them hard to use for domain scientists. Programming interfaces such as MPI are often too low-level for non-computer science experts and clutter up the application with a non-domain-relevant source code. On the other hand, there are high-level domain-specific languages or libraries that lack extensibility in order to support new scientific problems [4]. In addition, many of these solutions often lack the composability required for building libraries and integrating them seamlessly into larger applications, they deny an incremental approach that allows parallelizing an application step by step, or are limited to shared memory only. Therefore, users often resort to combining several of these solutions (e.g. MPI+OpenMP), which presupposes knowledge in at least two different programming models and entails a lack of resource management coordination that is left to the user.

In contrast, the AllScale API aims at providing the application developer with a single, extensible programming interface to express the parallel algorithms on a high level of abstraction, with automatic support for distributed memory.

The specific contributions of this work are:

- a shared-memory-style API for high-level specifications of algorithms and data structures with implicit distributed memory support,
- the capability of expressing new algorithms by extending the API with full compatibility to the rest of the software stack, and
- an evaluation of its programmability and performance using three real-life use cases.

While documentation and tutorials introducing the novice to the AllScale API are available online¹, the remainder of this work focuses on the API specification and important properties.

The rest of the paper is structured as follows. Section 2 discusses API design motivation while Section 3 and Section 4 detail API components. Implementation information is given in Section 5. Three real-world pilot applications and their respective API use are presented in Section 6 followed by an evaluation in Section 7. Related work is discussed in Section 8 and Section 9 provides the conclusion and future work.

¹ https://github.com/allscale/allscale_api/wiki

2 API DESIGN

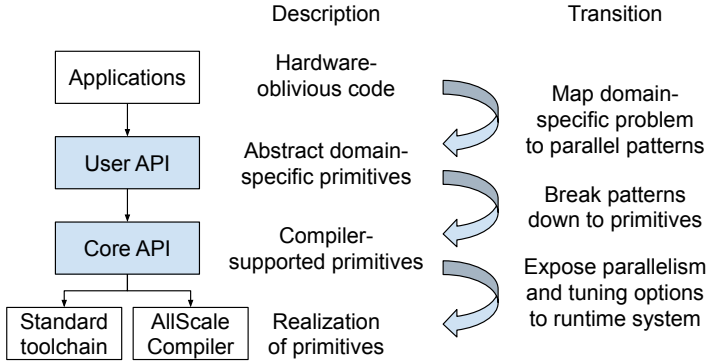


Figure 1. AllScale API design and usage overview

AllScale aims at providing domain scientists with the ability to write parallel applications for distributed memory using an API that is as easy to use as shared memory programming models such as OpenMP. While AllScale consists of many components including the API, a compiler, a distributed memory runtime system, and additional components for monitoring, resilience, etc., this work will present the API in detail. The overall architecture is discussed in detail by Jordan et al. [9].

The AllScale API is the façade of the AllScale Environment towards end-user applications. It provides the necessary primitives to express parallelism, data dependences, and needed synchronization steps within application code. The API is subdivided into two layers: the *AllScale Core API* and the *AllScale User API*. Their relationship is illustrated in Figure 1 and further discussed in the remainder of this section.

The Core API provides a concise set of basic generic primitives, comprising parallel control flow, synchronization, and communication constructs. It furthermore offers a generic data item interface that enables automatic data management of user-defined data structures. The User API is harnessing the expressive power of the Core API to provide specialized primitives for particular use cases, including basic constructs like parallel loops or adaptive grids.

The purpose of the subdivision into a Core and User API is to enable the implementation of a variety of parallel primitives on top of a small, concise set of central constructs which can be utilized to provide portability among different implementations of the AllScale Core API. Currently there are two implementations available within AllScale:

- a shared memory, pure C++ implementation, also referred to as the *standard toolchain*, which can be compiled by any C++14-compliant compiler with no further third-party library dependences – this implementation serves as a de-

velopment platform for AllScale applications and also represents a reference implementation; and

- the implementation utilizing the AllScale Compiler and Runtime System, also called the *AllScale toolchain*, which comprises a combination of static program analysis (crucial for automatically deriving data dependences required for distributed memory execution), code generation, scheduling, and resilience techniques to provide a highly scalable and portable implementation of the Core API on distributed memory systems.

Hence, applications developed within AllScale can be ported from shared to distributed memory simply by switching the toolchain, without any modifications required in the application. Additional parallel constructs may be introduced in the User API without the necessity of altering the underlying Core API implementation. Thus, the User API layer provides an effective way of extending the range of supported parallel patterns.

Furthermore, the User API shields application developers from the complexity of the Core API constructs. Due to the introduction of the User API efficient implementations of primitives native to the domain of the applications can be provided by parallelization experts. Therefore, AllScale provides a separation of concerns – with the overall task of providing efficient parallel codes – distributed among three contributors:

- the *domain expert*, aiming at obtaining the most effective algorithmic solution for the problem of interest;
- the *HPC expert*, able to develop efficient domain specific primitives to be used by the domain expert, focusing on e.g. communication and synchronization overheads or cache efficiency; and
- the *system-level expert* focusing on providing the most flexible and portable implementation of the Core API, hence handling load management, scheduling, resilience, and hardware management obligations.

The separation of responsibilities also effects the code base. By shielding the domain expert from all the underlying details (e.g. synchronization, communication, cache efficiency, scheduling, utilization of low-level parallel APIs), the resulting application code remains free of the otherwise necessary management code. This improves the maintainability of the resulting applications and thus the productivity of the domain expert.

3 CORE API

This section will detail the Core part of the AllScale API, specifically the primitives for parallel control flow and the concept of data items and their requirements. The User API, discussed in the section thereafter, builds on-top of these basic constructs to provide more high-level operations to domain experts. Note that while

the Core API also offers additional features such as a small performance profiling tool, discussing those exceeds the scope of this paper.

3.1 Parallel Control Flow

The AllScale Core API provides a single primitive for running concurrent tasks, resulting in feasible yet profound compiler and runtime system support for automatic distributed memory management of parallel applications. This single parallelism primitive forms the basis for all higher-level operators of the User API such as parallel loops, allowing the User API to be open for extension with new higher-level operators without any modifications required in the Core API or underlying compiler and runtime system [10].

This primitive, the *prec* [12] operator, is a higher order function combining three given functions into a new, recursive function. The three combined input functions are:

- a function testing for the base case of a recursion,
- a function processing the base case of a recursion, and
- a function processing the recursive step case.

The result is a new recursive function which, for a given input parameter, conducts the specified computation accordingly. To support an arbitrary input type, the *prec* operator has the type

$$\left(\begin{array}{l} \alpha \rightarrow \text{bool}, \\ \alpha \rightarrow \beta, \\ (\alpha, \alpha \rightarrow \text{treeure}\langle\beta\rangle) \rightarrow \text{treeure}\langle\beta\rangle \end{array} \right) \rightarrow (\alpha \rightarrow \text{treeure}\langle\beta\rangle)$$

where α is the parameter type of the resulting recursive function and $\text{treeure}\langle\beta\rangle$ is a parameterized abstract data type (ADT) modeling a handle on parallel tasks. The three parameters of the *prec* operator are the input functions discussed above. The resulting value of type $\alpha \rightarrow \text{treeure}\langle\beta\rangle$ is a function which, upon invocation, spawns a new task conducting the specified recursive operation in parallel. The resulting task handle can be utilized to orchestrate the parallel execution of additional tasks. A more in-depth discussion of ADTs can be found online [11].

3.2 Data Structure Primitives

While the parallel control flow primitive has been covered so far, it is not sufficient to compose parallel applications for distributed memory. In order to properly manage data dependences for parallel tasks executed in distinct memory address spaces, a specification for user-defined data structures needs to be defined as well. The purpose of this specification is to provide a single generic interface for HPC experts to implement new user-defined data structures while offering management access to the underlying runtime system for data distribution.

To this end, the data structure primitives offered by the Core API are a mere specification of any potential data type's interfaces and behaviors. Any data type T to be managed by an AllScale API implementation must provide a fragment type F for managing data storage and a range type R for addressing and managing sub-ranges of the data structure. Table 1 lists the operators required to be defined by F and R . Proper implementation of these operators for any arbitrary data structure ensures its suitability for automatic distributed data management by the AllScale Compiler and Runtime System. Several examples that implement widely-used data structures such as grids are discussed in Section 4.2 while their implementation, among others, can be found online².

Name	Type	Description
Fragment		
create	$R \rightarrow F$	creates a fragment covering (at least) the specified range
delete	$F \rightarrow \text{unit}$	deletes the given fragment
resize	$(F, R) \rightarrow \text{unit}$	alters the capacity of given fragment F to cover at least the range R
mask	$F \rightarrow T$	provides access to the data stored in fragment F via the interface defined by type T
extract	$(F, R) \rightarrow \text{Archive}$	extracts the data addressed by R from fragment F and packs it into an archive; <i>Archive</i> is a generic type of a utility provided by the API implementations to serialize data to be transferred between address spaces
insert	$(F, R, \text{Archive}) \rightarrow \text{unit}$	imports the data stored in the given archive into fragment F at the specified range R
Range		
union	$(R, R) \rightarrow R$	computes the union of two ranges
intersect	$(R, R) \rightarrow R$	computes the intersection of two ranges
difference	$(R, R) \rightarrow R$	computes the set difference of two ranges
empty	$(R) \rightarrow \text{bool}$	determines whether the given range is empty, thus addressing no elements
pack	$(R) \rightarrow \text{Archive}$	serializes instances
unpack	$(\text{Archive}) \rightarrow R$	deserializes instances

Table 1. Operators to be defined by fragment F and range R types of an AllScale data structure

3.3 IO Primitives

All sensible applications require input/output (IO) for their operations. While high-performance IO is a research topic on its own, the Core API offers basic primitives

² <https://git.io/fj4Xj>

to facilitate high-performance IO while keeping actual implementations abstract. To that end, the Core API provides two means for storage interaction:

- stream-based, providing unordered input and output facilities, facilitating e.g. the writing of simulation results to output streams,
- memory-mapped, providing read-only facilities for efficient random access within large data sets.

Their individual discussion in Section 3.3.1 and Section 3.3.2 illustrates the need for two separate components that match the different requirements while still providing efficient operations.

3.3.1 Stream-Based

The underlying concept of the AllScale stream-based IO interface is an out-of-order stream. Data entries can be atomically read from or written to such a stream. However, the order in which entries show up in the stream is undefined. Although tasks may be restricted due to imposed synchronization constraints to write data in a certain order to a stream pointing e.g. to a file, the resulting file may contain the written data in an arbitrary order. Furthermore, the API only guarantees the eventual visibility of a written element within an output stream, before the application terminates – not any particular timing. Thus, in particular, stream IO primitives may not be mis-used for implementing synchronization operations among tasks (nor are they required for this purpose for applications that adhere to the AllScale programming model).

Within the API we utilize the abstract types *istream* and *ostream* as a representation of an input or output stream. Table 2 lists the operations provided by stream-based IO.

Streams are designed to be the main facility to be utilized by application developers to produce output data without the artificial introduction of extra synchronization overhead. Furthermore, the abstraction to streams, their global addressing through names, and the lack of guarantees on the output order enables the flexible migration of tasks throughout the system. Tasks holding a stream to a file X on some node may be moved to another node, where they get assigned a new stream pointing to the logically same file. However, in reality the stream may point to a physically different output file maintained by the local runtime process. The concatenation of all the locally maintained output files controlled by the various AllScale Runtime System instances on a system are logically forming the actual output file. Thus, no synchronization beyond the boundaries of an AllScale node is required to facilitate streaming IO.

3.3.2 Memory-Mapped

In some cases, quite complex input data structures need to be handled. For instance, indexed files providing efficient access to desired sub-fractions may be loaded by

Name	Type	Description
read	$(istream) \rightarrow \alpha$	atomically reads an element of type α from the given input stream
atomic	$\left(\begin{array}{c} istream, \\ (istream) \rightarrow unit \end{array} \right) \rightarrow unit$	An operator providing atomic access to an input stream, enabling the provided function to read a sequence of consecutive elements in order
write	$(ostream, \alpha) \rightarrow unit$	atomically writes the given element of type α to the given output stream, where it will be visible eventually
atomic	$\left(\begin{array}{c} ostream, \\ (ostream) \rightarrow unit \end{array} \right) \rightarrow unit$	An operator providing atomic access to an output stream, enabling the provided function to write a sequence of consecutive elements in order
create_in	$(string) \rightarrow istream$	opens an input file with the given name and provides a stream to read from it; the file format is implementation-specific and data may only be read and written using the AllScale IO API
create_out	$(string) \rightarrow ostream$	creates a new empty file under the given name and provides an output stream to write information to the file; the file format is implementation specific and may only be read using AllScale IO primitives
get_in	$(string) \rightarrow istream$	obtains an input stream to a previously opened input file which might be concurrently read
get_out	$(string) \rightarrow ostream$	obtains an output stream of a previously opened output file which might be concurrently written to

Table 2. Operations supported by stream-based IO

an application. Since the sequential access through streams would impose a major performance penalty for accessing such files, memory-mapped IO is offered for read only files. It provides the means for efficient random read-only access of sub-sets of larger data, with open files available in the address spaces of all AllScale runtime system processes.

The abstract type referencing a memory-mapped IO file is *mmfile*. Table 3 lists the operations provided by memory-mapped IO.

Opening and closing memory-mapped files is a global operation throughout the system. Once a file is opened, it is available within the address spaces of all runtime system processes, although not necessarily at the same address range. The task

Name	Type	Description
open	$(string) \rightarrow mmfile$	globally opens a memory-mapped file with the given path
get	$(string) \rightarrow mmfile$	obtains a reference to a previously opened memory-mapped file
access	$(mmfile) \rightarrow \alpha$	interprets the content of the memory-mapped file as a value of type α
close	$(mmfile) \rightarrow unit$	globally closes a memory-mapped file such that it is no longer available for any process in the application

Table 3. Operations supported by memory-mapped IO

migration of the runtime system ensures that references to such files are adapted accordingly whenever a task is migrated between nodes.

Memory-mapped IO is mainly considered a facility for special use cases in the construction of efficient data structures within the User API layer. An example is the static graph structure of a mesh (see Section 4.2.3). While it might also be utilized by the end user, it will always be strictly limited to read-only use cases. Write operations are restricted to the steam-based IO API.

4 USER API

The generic nature of the Core API exceeds the complexity which could be effectively handled by domain experts for implementing parallel algorithms. For this reason, the AllScale User API aims at providing a set of more user-friendly, higher-level constructs for the composition of parallel applications by domain experts. The implementation of these constructs is carried out by HPC experts utilizing the primitives offered by the Core API.

4.1 Parallel Control Flow Constructs

While the User API is open for extension with new parallel patterns as required, several frequently-occurring patterns such as parallel loops are already provided and discussed below.

4.1.1 Parallel Loops

A vast majority of algorithms expressing data parallelism rely on parallel loops. They provide the means to perform computational work in an iteration space in parallel at the cost of executing the individual iterations concurrently and in an arbitrary order. To that end, the User API offers a parallel loop construct for realizing data-parallel programming within the AllScale environment.

Let *iterator* be a random access iterator. Then the *pfor* operator provides a parallel loop execution with the parameters defined in Table 4. Figure 2 shows a sample usage of the *pfor* operator with fine-grained synchronization. Several of these syn-

chronization patterns are available, such as *neighborhood_sync* or *one_on_one*. HPC experts are free to extend these by new patterns not yet covered.

Name	Type	Description
begin	<i>iterator</i>	inclusive beginning of the iterator range
end	<i>iterator</i>	exclusive end of the iterator range
body	$(\text{iterator}) \rightarrow \beta$	the function applied to each element
dependence	$\text{dep}(\text{iterator})$	optional dependence for fine-grained synchronization

Table 4. Parameters of the *pfor* operator

```

1      #include <array>
2      #include <allscale/api/user/algorithm/pfor.h>
3      namespace alg = allscale::api::user::algorithm;
4      using ArrayType = std::array<int,N>;
5      const int N = 200;
6      void initAndIncrement(const ArrayType& data, ArrayType& output) {
7      auto ref = alg::pfor(0,N,[&](int i) {
8          output[i] = ...; // initialization
9      });
10     alg::pfor(1,N-1,[&](int i) {
11         output[i] += data[i+1] + data[i] + data[i-1];
12     }, alg::neighborhood_sync(ref));
13 }
```

Figure 2. Two *pfor* operators initializing and incrementing data in a `std::array` with fine-grained synchronization. The second *pfor* will execute iteration i after the first has finished its iterations $i-1$, i , and $i+1$. Constructs specific to the AllScale API are shown in blue and underlined.

4.1.2 Recursive Space/Time Decomposition

A frequently utilized template for large-scale high-performance applications are stencils. In a stencil-based application, an update operation is iteratively applied to the elements of an n -dimensional array of cells. Thereby, for each update, the update operation is combining the previous values of cells within a locally confined area surrounding the targeted cell location to obtain the updated value for the targeted cell. Since these update operations within a single update step (also known as timestep) are independent, this application pattern provides a valuable source for parallelism within a correspondingly shaped application. The User API offers the *stencil* operator, the parameters of which are defined in Table 5.

Name	Type	Description
timesteps	int	number of time steps to be computed
size	int^n	spatial size of n -dimensional data to be processed
kernel function	$(int, int^n, \alpha^{s_1 \times \dots \times s_n}) \rightarrow \alpha$	update function accepting current time, location, and grid, computing the resulting value
kernel shape	$(int^n)^*$	compile-time-constant list of offsets to cells accessed by kernel, determining its shape
boundary function	$(int, int^n, \alpha^{s_1 \times \dots \times s_n}) \rightarrow \alpha$	update function for boundary cases, where some elements are outside the grid
initialization function	$(int^n) \rightarrow \alpha$	computes initial value for cell at given coordinate
finalization function	$(int^n, \alpha) \rightarrow unit$	function consuming value of cell at the end of a computation
observers	$\left(\begin{array}{l} (int, int^n) \rightarrow bool, \\ (int, int^n, \alpha) \rightarrow unit \end{array} \right)^*$	list of pairs, each describing an observer with time/location filtering function and actual trigger function to be applied

Table 5. Parameters of the *stencil* operator

4.1.3 Additional Operations

Beyond the *pfor* and *stencil* operators presented thus far, the User API offers additional parallel operations that are frequently encountered in parallel applications. These include e.g. the *map-reduce* operator for data aggregation, the *async* operator for single tasks, or the *vcycle* operator for multi-grid methods. However, a more detailed presentation is omitted for brevity.

4.2 Data Structures

4.2.1 Grid

A frequently-encountered data structure in high-performance codes is formed by n -dimensional arrays of values. While many programming languages support such structures for arbitrary dimensions, C/C++ only supports one-dimensional, dynamically sized arrays natively. However, this leaves creation and management of these structures to the user, forming a major obstacle for the usability of C++ on distributed memory systems.

To ease the use of C++ for use cases depending on such structures, the AllScale User API provides a uniform *Grid* data structure providing the following features:

- regular n -dimensional array of runtime-defined size,
- efficient read/write random access operators,
- efficient scan operation (processing all elements),
- type-parameterized in element type and no. of dimensions,
- enforces the serializability of its element types,
- implements data item concept for automated distribution.

Let $Grid\langle\alpha, n\rangle$ be the abstract data type family implemented by the AllScale User API to represent n -dimensional grids, where α is a type variable specifying the element type. Furthermore, let $type\langle\alpha\rangle$ be the meta type of type α . Then Table 6 lists the operators defined on $Grid$ data structures.

Name	Type	Description
create	$\left(\begin{array}{c} type\langle\alpha\rangle, \\ int^n \end{array} \right) \rightarrow Grid\langle\alpha, n\rangle$	creates new n -dimensional grid with element type α of given size
destroy	$(Grid\langle\alpha, n\rangle) \rightarrow unit$	deletes given grid
read	$\left(\begin{array}{c} Grid\langle\alpha, n\rangle, \\ int^n \end{array} \right) \rightarrow \alpha$	reads element from given grid at specified coordinates
write	$\left(\begin{array}{c} Grid\langle\alpha, n\rangle, \\ int^n, \\ \alpha \end{array} \right) \rightarrow unit$	updates element within given grid at specified coordinates
scan	$\left(\begin{array}{c} int^n, \\ int^n, \\ (int^n) \rightarrow \beta \end{array} \right) \rightarrow treecture\langle unit \rangle$	applies given function (in parallel) to all elements of given interval in arbitrary order

Table 6. Operators defined on $Grid$ data structures

Figure 3 illustrates the use of such a $Grid$ data structure. In this case, $Grid\langle int, 2 \rangle$ (type T , as described in Section 3.2) offers operators for accessing elements within a two-dimensional structure, indexed by coordinates of type $GridRegion$ (line 7, the type is not explicitly visible in this example code). $GridRegion$ is the corresponding range type R of T and holds a conjunction of 2D-coordinate pairs describing axis-aligned boxes covering the range to be described – in this case a single point at position $\{7, 9\}$. An instance of type $GridFragment\langle double, 2 \rangle$ (the corresponding fragment type F of T , generally not visible in user code) realizes the actual storage of fragments of the data stored in $Grid$ instances; the implementation may hold a reference to allocated memory plus the coordinates of the covered ranges. For further reference, the $Grid$ implementation of types T , R and F is available online³.

³ <https://git.io/JUC1F>

```

1      #include <allscale/api/user/data/grid.h>
2      // create a two-dimensional grid of integers of size 10x20
3      allscale::api::user::data::Grid<int, 2> grid({10,20});
4      // initialize all elements with 1.0
5      grid.pforEach([](int& element) { element = 1.0; });
6      // set element at position [7,9] to 5.0
7      grid[{7,9}] = 5.0;

```

Figure 3. Example usage of the *Grid* data structure

4.2.2 Adaptive Grid

The *Adaptive Grid* is an advanced variant of the *Grid* structure also frequently encountered within a simulation code. In addition to the properties of *Grid*, the *Adaptive Grid* provides means to nest grids within grid cells. For a given instance, each top-level grid cell contains an identically structured fixed-length sequence of grids. The first of those contains a single cell. Every consecutive grid contains a multiple number of cells per dimension of its predecessor. Each top-level grid cell comprising the sequence of its nested grids is referred to as an *Adaptive Grid Cell*.

Let $AGrid\langle\alpha, n, [r_1, \dots, r_l]\rangle$ be the ADT family implemented by the AllScale User API to represent n -dimensional *Adaptive Grids*, where α is a type variable specifying the element type, r_1, \dots, r_l the refinement factors, and l the number of refinement levels. Thus, the size of the grid at level i is defined by

$$s(i) = \begin{cases} [1, \dots, 1] \in int^n, & \text{if } i = 0, \\ s(i-1) * r_i, & \text{otherwise.} \end{cases}$$

To address elements within an *Adaptive Grid* an extension of *Grid* coordinates is required. While elements within a *Grid* can be addressed using a single coordinate of type int^n , the *Adaptive Grid* requires information regarding the location of the addressed element in the nested grid structure. Thus, additional coordinates to navigate through these refinement layers are required. Hence, to address an element within an *Adaptive Grid*, a hierarchical coordinate of type $(int^n)^+$ is required. For instance, the coordinate $[[7, 3], [2, 4], [8, 2]]$ addresses the element located within the cell that can be reached by navigating first to the top-level cell $[7, 3]$, continuing to cell $[2, 4]$ of its first refinement layer, and ending up within cell $[8, 2]$ of the second refinement layer. Let $seq\langle r_1, \dots, r_l \rangle$ be the static meta-type of a sequence of integers r_1, \dots, r_l , then Table 7 lists the operators defined on *Adaptive Grid* data structures.

4.2.3 Unstructured Mesh

The *Mesh* data structure is designed to represent a graph structure of multiple node types that are connected through various types of edges. Furthermore, a *Mesh* may

Name	Type	Description
create	$\left(\begin{array}{c} type\langle\alpha\rangle, \\ int^n, \\ seq\langle r_1, \dots, r_l \rangle \end{array} \right) \rightarrow AGrid\langle\alpha, n, [r_1, \dots, r_l]\rangle$	creates new n -dimensional adaptive grid with element type α of given size and grid cell structure
destroy	$(AGrid\langle\alpha, n, [r_1, \dots, r_l]\rangle) \rightarrow unit$	deletes the given adaptive grid
read	$\left(\begin{array}{c} AGrid\langle\alpha, n, [r_1, \dots, r_l]\rangle, \\ (int^n)^+ \end{array} \right) \rightarrow \alpha$	reads element from given grid at specified hierarchical coordinates
write	$\left(\begin{array}{c} AGrid\langle\alpha, n, [r_1, \dots, r_l]\rangle, \\ (int^n)^+, \\ \alpha \end{array} \right) \rightarrow unit$	updates element within given grid at specified hierarchical coordinates
refine	$\left(\begin{array}{c} AGrid\langle\alpha, n, [r_1, \dots, r_l]\rangle, \\ (int^n)^+, \\ Grid\langle\alpha, n\rangle \end{array} \right) \rightarrow unit$	refines resolution of cell addressed by given hierarchical coordinate by inserting given grid data as refinement information
coarsen	$\left(\begin{array}{c} AGrid\langle\alpha, n, [r_1, \dots, r_l]\rangle, \\ (int^n)^+, \\ \alpha \end{array} \right) \rightarrow unit$	coarsens resolution of cell addressed by given hierarchical coordinate and inserting given value data as coarsened information
getLevel	$\left(\begin{array}{c} AGrid\langle\alpha, n, [r_1, \dots, r_l]\rangle, \\ (int^n)^+ \end{array} \right) \rightarrow int$	gets currently active resolution level at a specified hierarchical grid position
scan	$\left(\begin{array}{c} int^n, \\ int^n, \\ AGrid\langle\alpha, n, [r_1, \dots, r_l]\rangle, \\ ((int^n)^+) \rightarrow \beta \end{array} \right) \rightarrow treecture\langle unit \rangle$	applies given function (in parallel) to all active hierarchical coordinates of a given interval in an arbitrary order

Table 7. Operators defined on *Adaptive Grid* data structures

consist of several layers, which describe the same graph in different levels of detail. Hierarchical edges may connect the same nodes of different layers.

Besides the topological information maintained by *Mesh* instances, means to maintain attributes associated to nodes, edges, and hierarchical edges within a *Mesh* need to be included. For instance, node IDs, coordinates, volumes, temperatures, and other domain space specific properties may be incorporated through this facility.

Let n_1, \dots, n_m be a list of node types, $e_1, \dots, e_k \in \{n_1, \dots, n_m\}^2$ a list of edge types, and $h_1, \dots, h_o \in \{n_1, \dots, n_m\}^2$ a list of hierarchical edge types. Then the type $Mesh\langle [n_1, \dots, n_m], [e_1, \dots, e_k], [h_1, \dots, h_o], l \rangle$ represents the type of a *Mesh* structure including the given node, edge, and hierarchical edge types on l layers. Furthermore, let $id\langle \alpha, l \rangle$ be an identifier for an element of type α on layer l within a *Mesh* – thus the type of ID used for addressing nodes, edges, or hierarchical edges within meshes. Also, let $MData\langle n, l, \alpha \rangle$ be the type of an attribute collection associating values of type α to nodes of type n located on layer l of some *Mesh* instance. Finally, let $MBuilder\langle [n_1, \dots, n_m], [e_1, \dots, e_k], [h_1, \dots, h_o], l \rangle$ be the type of construction utility for creating meshes. Then Table 8 lists the operations defined on these types.

5 IMPLEMENTATION

The AllScale API is based on C++, which allows the re-use of existing tools such as debuggers, and makes heavy use of template-based meta-programming. This built-in language feature of C++ enables the scripted generation of code during compilation. Widely utilized examples include the generation of data structures like vectors, sets, or maps specialized to specific type parameters. However, the capabilities of this feature reach much further. It also enables the generic implementation of primitives, where a single primitive may cover a wide range of use cases, without the introduction of any abstraction overhead. All primitives of the AllScale Core API are generic primitives, making heavy use of C++ meta-programming features for the automated synthetization of program code. The same applies for all AllScale User API constructs, to improve their (re-)usability and flexibility.

In addition, the standard toolchain implementation of the API only requires a C++14-compliant compiler and standard library (e.g. recent versions of GCC, Clang, Apple-Clang, and Visual Studio), and hence supports application development on at least three different operating systems (Linux, OS X, Windows). In order to mitigate the initial adoption barrier of porting applications to AllScale, an SDK comprising a build system infrastructure and setup scripts is provided⁴.

6 USE CASES

This section presents our real-world pilot applications that build on the AllScale API. The first, iPIC3D [14], is a particle-in-cell simulation code developed together with KTH Stockholm and employs multiple *pfor* operators and 3-dimensional *Grid*

⁴ https://github.com/allscale/allscale_sdk

Name	Type	Description
Mesh Builder		
create	$(type\langle Mesh\langle n, e, h, l \rangle \rangle) \rightarrow MBuilder\langle n, e, h, l \rangle$	creates builder for a given mesh type; initially, mesh is empty
destroy	$(MBuilder\langle n, e, h, l \rangle) \rightarrow unit$	destroys builder instance
addNode	$\left(\begin{array}{c} MBuilder\langle [n_1, \dots, n_m], e, h, l \rangle, \\ type\langle n \rangle, \\ type\langle i \rangle \end{array} \right) \rightarrow id\langle n, i \rangle$	creates a new node of a given type n on a given level i within mesh under construction
link	$\left(\begin{array}{c} MBuilder\langle n, [(n_{1a}, n_{1b}), \dots, (n_{ka}, n_{kb})], h, l \rangle, \\ id\langle n_{ia}, j \rangle, \\ id\langle n_{ib}, j \rangle \end{array} \right) \rightarrow unit$	ads edge to mesh under construction
link	$\left(\begin{array}{c} MBuilder\langle n, e, [(n_{1a}, n_{1b}), \dots, (n_{oa}, n_{ob})], l \rangle, \\ id\langle n_{ia}, j + 1 \rangle, \\ id\langle n_{ib}, j \rangle \end{array} \right) \rightarrow unit$	adds hierarchical edge to mesh under construction
toMesh	$(MBuilder\langle n, e, h, l \rangle) \rightarrow Mesh\langle n, e, h, l \rangle$	obtains a copy of mesh under construction
Mesh Structure		
store	$(Mesh\langle n, e, h, l \rangle) \rightarrow byte^*$	serializes mesh
load	$\left(\begin{array}{c} byte^*, \\ type\langle Mesh\langle n, e, h, l \rangle \rangle \end{array} \right) \rightarrow Mesh\langle n, e, h, l \rangle$	deserializes given byte array to mesh
destroy	$(Mesh\langle n, e, h, l \rangle) \rightarrow unit$	destroys given mesh
getNeighbors	$\left(\begin{array}{c} Mesh\langle n, [(n_{1a}, n_{1b}), \dots, (n_{ka}, n_{kb})], h, l \rangle, \\ type\langle n_{ia}, n_{ib} \rangle, \\ id\langle n_{ia}, j \rangle \end{array} \right) \rightarrow id\langle n_{ib}, j \rangle^*$	obtains a list of neighbors of the given node following given kind of edge
getParents	$\left(\begin{array}{c} Mesh\langle n, e, [(h_{1a}, h_{1b}), \dots, (h_{oa}, h_{ob})], l \rangle, \\ type\langle h_{ia}, h_{ib} \rangle, \\ id\langle h_{ia}, j \rangle \end{array} \right) \rightarrow id\langle h_{ib}, j + 1 \rangle^*$	obtains a list of parents of the given node following given kind of hierarchical edge
getChildren	$\left(\begin{array}{c} Mesh\langle n, e, [(h_{1a}, h_{1b}), \dots, (h_{oa}, h_{ob})], l \rangle, \\ type\langle h_{ia}, h_{ib} \rangle, \\ id\langle h_{ib}, j \rangle \end{array} \right) \rightarrow id\langle h_{ia}, j - 1 \rangle^*$	obtains a list of children of the given node following given kind of hierarchical edge
scan	$\left(\begin{array}{c} Mesh\langle [n_1, \dots, n_m], e, h, l \rangle, \\ type\langle n_i \rangle, \\ type\langle j \rangle, \\ (id\langle n_i, j \rangle) \rightarrow \beta \end{array} \right) \rightarrow treecture\langle unit \rangle$	applies given operation to every instance of selected node type on selected level within given mesh
scan	$\left(\begin{array}{c} Mesh\langle n, [(n_{1a}, n_{1b}), \dots, (n_{ka}, n_{kb})], h, l \rangle, \\ type\langle (n_{ia}, n_{ib}) \rangle, \\ type\langle j \rangle, \\ (id\langle n_{ia}, j \rangle, id\langle n_{ib}, j \rangle) \rightarrow \beta \end{array} \right) \rightarrow treecture\langle unit \rangle$	applies given operation to every instance of selected edge type on selected level within given mesh
scan	$\left(\begin{array}{c} Mesh\langle n, e, [(n_{1a}, n_{1b}), \dots, (n_{oa}, n_{ob})], l \rangle, \\ type\langle (n_{ia}, n_{ib}) \rangle, \\ type\langle j \rangle, \\ (id\langle n_{ia}, j + 1 \rangle, id\langle n_{ib}, j \rangle) \rightarrow \beta \end{array} \right) \rightarrow treecture\langle unit \rangle$	applies given operation to every instance of selected hierarchical edge type on selected pair of adjacent levels within given mesh

Mesh Data		
create	$\left(\begin{array}{c} Mesh\langle n_1, \dots, n_m, e, h, l \rangle, \\ type\langle n_i \rangle, \\ type\langle j \rangle, \\ type\langle \alpha \rangle \end{array} \right) \rightarrow MData\langle n_i, j, \alpha \rangle$	creates attribute storage associating value of type α to each node instance of type n_i on layer j present in given mesh
destroy	$(MData\langle n, l, \alpha \rangle) \rightarrow unit$	deletes attribute storage
read	$\left(\begin{array}{c} MData\langle n, l, \alpha \rangle, \\ id\langle n, l \rangle \end{array} \right) \rightarrow \alpha$	retrieves value of attribute associated to given node from given attribute store
write	$\left(\begin{array}{c} MData\langle n, l, \alpha \rangle, \\ id\langle n, l \rangle, \\ \alpha \end{array} \right) \rightarrow unit$	updates value of attribute associated to given node in given attribute store

Table 8. Operators defined on *Mesh* builder, *Mesh* structure and *Mesh* data

data structures. The second, AMDADOS [2], is an advection-diffusion code developed together with IBM Research Ireland and uses a 2-dimensional *stencil* operator and *adaptive grid* structure. While the full implementation of these applications is available online with an in-depth discussion available in literature [17], we only present code excerpts of the main computation here for brevity.

6.1 iPIC3D

The iPIC3D pilot application is an iterative particle-in-cell space weather simulation code and its main computation loop is shown in Figure 4. Its underlying data structure is a 3-dimensional regular equidistant *Grid* (line 13) where each element is a cell representing a cuboid and maintaining a dynamically-sized list of particles (line 8) located in this cuboid. Furthermore, each particle stores physical properties such as location, velocity, charge, and mass.

In each iteration of the simulation, the physical effects of the particles are aggregated to compute a set of induced force fields (lines 21–26). These force fields are also represented by 3-dimensional *Grid* structures (lines 9 and 14). In a next step, electromagnetic field equations are solved (lines 27–29), the forces affecting each particle's position and velocity are computed and the particles are updated accordingly (lines 35–37). Particles moving beyond the boundary of a cell need to be migrated (lines 33–35) to the respective target cell, which can be any of 26 neighbor cells. Once the migration of particles is completed, the next iteration can be computed.

The simulation is set up such that particles may never move fast enough to skip a full cell over the duration of a single time step (= iteration step). This property is effectively restricting communication patterns, such that e.g. regions that are n cells apart may differ in their simulation time by up to n time steps. It also localizes communication since particles may only be exchanged between adjacent cells.

```

1      unsigned numSteps = ...; // number of time steps
2      auto zero = utils::Coordinate<3>(0); // point of origin
3      auto size = ...; // size of domain
4
5      namespace alg = allscale::api::user::algorithm;
6      namespace data = allscale::api::user::data;
7
8      struct Cell { std::vector<Particle> particles; };
9      struct FieldNode { ... // electric and magnetic field components };
10     struct DensityNode { Vector3<double> J; // current density };
11
12     // 3D grids for cells, electromagnetic field and current density
13     data::Grid<Cell, 3> cells = ...;
14     data::Grid<FieldNode, 3> field = ...;
15     data::Grid<DensityNode, 3> density = ...;
16     // create a grid of buffers for density projection from particles to grid nodes
17     data::Grid<DensityNode> densityContributions(size * 2);
18
19     // run time loop for the simulation
20     for(unsigned i = 0; i < numSteps; ++i) {
21         alg::pfor(zero, size, [&](const utils::Coordinate<3>& pos) {
22             //STEP 1a: collect particle density contributions and store in
23             //buffers
24             particleToFieldProjector(cells[pos], pos, densityContributions); });
25         alg::pfor(zero, density.size(), [&](const utils::Coordinate<3>& pos) {
26             // STEP 1b: aggregate density in buffers to density nodes
27             aggregateDensityContributions(densityContributions, pos, density[
28                 pos]); });
29         alg::pfor(fieldStart, fieldEnd, [&](const utils::Coordinate<3>& pos){
30             // STEP 2: solve electromagnetic field equations
31             fieldSolver(pos, density, field); });
32         alg::pfor(zero, size, [&](const utils::Coordinate<3>& pos){
33             // STEP 3: project forces to particles and move particles
34             particleMover(cells[pos], pos, field, particleTransfers); });
35         alg::pfor(zero, size, [&](const utils::Coordinate<3>& pos){
36             // STEP 4: transfer particles into destination cells
37             transferParticles(cells[pos], pos, particleTransfers); });
38     }

```

Figure 4. Code excerpt of main data structures and simulation loop of iPIC3D. The full code is available online at https://github.com/allscale/allscale_ipic3d.

These major simulation steps are all parallel update operations using higher-dimensional variations of the *pfor* operator. Thus, the resulting simulation code is structured like a list of update loops, enclosed within a single time step loop.

A crucial step for distributed memory implementations is the exchange of particles and field strength values among adjacent cells. The original iPIC3D implementation requires three synchronization steps for each time step based on explicit halo cell updates. The AllScale Toolchain integrates those exchanges implicitly, transparent to the application developer, and overlaps computation and communication to reduce the impact of necessary synchronization steps.

6.2 AMDADOS

AMDADOS is a numerical simulation of an oil spill, with an excerpt of the main computation code shown in Figure 5. It is based on a 2-dimensional stencil (lines 20–44) and incorporates data assimilation events (line 28) using external sensor data (line 16) in order to mitigate simulation errors. The basic data structure of this application is a regular, *Adaptive Grid* (line 18). The number of refinement levels is a compile-time constant and can be hard coded within the application (lines 4–8). However, coarsening and refinement steps are applied dynamically during execution based on the state of the simulation as well as data assimilation events (inside the functions called in lines 26 and 28, not shown in detail here).

The resolution refinement follows a hierarchical pattern. On the top level, a fixed size, regular 2D grid defines the domain of the overall simulated area. Each of these top-level cells (also called sub-domains) may then be itself recursively subdivided into small regular grids, up to a statically fixed maximum resolution. The simulation algorithm updates each sub-domain independently for a single time step at the currently active level of resolution. This update operation may take several iterations, yet does not necessitate the exchange of any information with neighboring sub-domains. Once complete, boundary information is exchanged between adjacent sub-domains. Thus, sub-domains being n top-level cell-widths apart may be n time steps apart in their simulation time.

While this application could be implemented using the *pfor* operator, this would lead to a flat parallelism structure with synchronization enforced at the end of each time step. For this reason, it utilizes the *stencil* operator instead, which exposes recursive space-time decomposition and allows multiple time steps on spatially sufficiently separated sub-domains to be computed in parallel – sub-domains being n global cell-widths apart may be n time steps apart in their simulation time. In addition, it shows the observer functionality of the *stencil* operator, which allows for time- and space-controlled output of the simulation state.

The assimilation of data (line 28) is an optional step after the completion of an update of a sub-domain. In this case, the solution obtained for the processed sub-domain is combined with some externally obtained measurement before the simulation continues with the mutual exchange of information among adjacent cells and the next simulation time step.

```

1      namespace alg = algorithm;
2      namespace alg = data;
3
4      typedef data::CellConfig<2, data::layers<
5          data::layer<1,1>, // layer 2, size 1 x 1
6          data::layer<8,8>, // layer 1, size 8 x 8
7          data::layer<2,2> // layer 0, size 16 x 16
8      >> subdomain_config_t;
9
10     using subdomain_t = data::AdaptiveGridCell<double, subdomain_config_t
11         >;
12     using domain_t = data::Grid<subdomain_t, 2>;
13
14     struct Sensor { ... };
15
16     // 2D grid of sensor data
17     const data::Grid<Sensor, 2> sensors = ...;
18     const size_t Nt = ...; // number of time steps
19     domain_t state_field = ...; // 2D grid of sub-domains constitutes entire
20     domain
21
22     alg::stencil(state_field, Nt,
23     [&,Nt](time_t t, const point2d_t& idx, const domain_t& state)
24     -> const subdomain_t
25     { // Computation of subdomains
26         subdomain_t temp_field;
27         if(contexts[idx].sensors.empty()) // subdomain without sensors
28             SubdomNoSensors(t, state, temp_field, contexts[idx], idx);
29         else // subdomain with sensors, assimilation occurs
30             SubdomKalman(sensors[idx], t, state, temp_field, contexts[idx], idx);
31         return temp_field;
32     },
33     alg::observer( // Monitoring: periodically write full subdomain to file
34         [=](time_t t) { return (t % output_interval == 0); // time filter },
35         [](const point2d_t&) { return true; // Space filter: no constraints
36             },
37         // Append a full field to the file of simulation results.
38         [&](time_t t, const point2d_t& idx, const subdomain_t& cell) {
39             cell.forAllActiveNodes([&](const point2d_t& loc, double val) {
40                 point2d_t glo = computeGlobalIndex(loc, idx);
41                 out_stream.atomic([=](auto& file) {
42                     file << ... << "\n"; // write output data
43                 });
44             });
45         });

```

```

41         });
42     }
43 )
44 );

```

Figure 5. Code excerpt of the main stencil computation of AMDADOS. The full code is available online at https://github.com/allscale/allscale_amdados.

An assimilation operation, however, is orders of magnitudes more complex than a mere simulation time step for the same sub-domain. These trigger load imbalance that has to be dealt with. Thus the application consists of fixed-size subdomains with two levels of varying computational expenses:

- coarsely resolved sub-domains incur less computational workload than finer resolved sub-domains,
- optional data assimilation steps on sub-domains incur significantly additional, sporadic computational costs.

6.3 FINE/Open

The FINE/Open application is a computational fluid dynamics (CFD) solver developed by NUMECA [16]. The underlying data structure is a static, unstructured *Mesh* comprising objects such as cells, faces, edges, nodes, or boundary faces. The computation is based on a *vcycle*, which is further detailed below.

While it is not possible to show the actual implementation of FINE/Open due to non-disclosure concerns, Figure 6 shows a basic code example which is built on-top of the same *Mesh* data structure and also performs computations using the *vcycle* operator. The geometric information is covered by a list of relations connecting cells, faces and vertices (line 3 and lines 5–9) with each other (e.g. a relation of a cell to its faces). These relations can be easily navigated by templated member functions (lines 45–46, 51 and 53). Note that the type system of C++ ensures proper object navigation during compile-time (e.g., attempting to illegally access a vertex from a face would result in a compiler error). Furthermore, for each object, a set of properties influencing the simulation is maintained (lines 13–16). These may comprise static information like e.g. the volume of a cell or dynamic information such as the heat flow through a face. The latter is the state of the conducted simulation and the result end users are interested in. Finally, to aid the effective computation of the desired solution, multiple meshes describing the same objects in different resolutions are combined into a hierarchy of meshes to enable the use of multi-grid solvers (lines 39–60). The hierarchy of meshes can be navigated via hierarchical edges (line 11).

In each simulation step, updates to the various properties associated to the mesh objects are conducted. Updates start in the mesh layer exhibiting the finest resolution. Thereby, physical effects are propagated through the connections between the


```

1      using namespace allscale::api::user;
2      // -- elements --
3      struct Cell {}; struct Face {}; struct Vertex {};
4      // -- edges --
5      struct Cell2Vertex : public data::edge<Cell, Vertex> {};
6      struct Cell2Face_In : public data::edge<Cell, Face> {};
7      struct Cell2Face_Out : public data::edge<Cell, Face> {};
8      struct Face2Cell_In : public data::edge<Face, Cell> {};
9      struct Face2Cell_Out : public data::edge<Face, Cell> {};
10     // -- hierarchical edges --
11     struct Parent2Child : public data::hierarchy<Cell,Cell> {};
12     // -- property data --
13     struct CellTemperature : public data::mesh_property<Cell,double> {};
14     struct FaceArea : public data::mesh_property<Face,value_t> {};
15     struct FaceVolRatio : public data::mesh_property<Face,value_t> {};
16     struct FaceConductivity : public data::mesh_property<Face,double> {};
17     // -- define the mesh and builder --
18     template<unsigned levels = 1>
19     using MeshBuilder = data::MeshBuilder<
20     data::nodes<Cell, Face, Vertex>,
21     data::edges<Cell2Vertex, Cell2Face_In, Cell2Face_Out, Face2Cell_In,
22     Face2Cell_Out>,
23     data::hierarchies<Parent2Child>,
24     levels>;
25     // -- type of a mesh --
26     template<unsigned levels = 1, unsigned PDepth = P_DEPTH>
27     using Mesh = typename MeshBuilder<levels>::template mesh_type<
28     PDepth>;
29     // -- type of the properties of a mesh --
30     template<typename Mesh>
31     using MeshProperties = data::MeshProperties<Mesh::levels,
32     typename Mesh::partition_tree_type, CellTemperature, FaceConductivity,
33     VertexPosition>;
34     // V-Cycle stage
35     template<typename Mesh, unsigned Lvl>
36     struct TemperatureStage {
37     const Mesh& mesh; // mesh structure, properties and cell data
38     MeshProperties<Mesh>& properties;
39     attribute<Cell, value_t> temperature;
40     attribute<Face, value_t> fluxes;
41
42     void jacobiSolver() {
43         auto& fConductivity = properties.template get<FaceConductivity
44         , Lvl>();
45         auto& fArea = properties.template get<FaceArea, Lvl>();

```

```

42      auto& fVolumeRatio = properties.template get<FaceVolRatio,
43          Lvl>();
44
45      mesh.template pforAll<Face, Lvl>([&](auto f) { // compute per-
46          face flux
47          auto in = mesh.template getNeighbor<Face2Cell.In>(f);
48          auto out = mesh.template getNeighbor<Face2Cell.Out>(f);
49          value_t gradTemp = temperature[in] - temperature[out];
50          fluxes[f] = fVolumeRatio[f] * fConductivity[f] * fArea[f] * gradTemp;
51          });
52      mesh.template pforAll<Cell, Lvl>([&](auto c) { // update per-
53          cell solution
54          auto subtractingFaces = mesh.template getNeighbors<
55              Face2Cell.In>(c);
56          for(auto sf : subtractingFaces) { temperature[c] -= fluxes[sf]; }
57          auto addingFaces = mesh.template getNeighbors<Face2Cell.Out
58              >(c);
59          for(auto af : addingFaces) { temperature[c] += fluxes[af]; }
60          }); }
61
62      void computeFineToCoarse() { ... }
63      void computeCoarseToFine() { ... }
64      void restrictFrom(TemperatureStage<Mesh,Lvl-1>& childStage) { ... }
65      void prolongateTo(TemperatureStage<Mesh,Lvl-1>& childStage) { ... }
66  };

```

Figure 6. Code excerpt of a sample application using the *vcycle* operator on a multi-grid mesh. The full version is available online at <https://git.io/fjBTq>.

various objects on this layer. After a fixed number of iterations, the current state of the simulated properties are aggregated and projected to the next coarser-grained level of the hierarchical mesh. There, the same propagation and aggregation operations are repeated. After completing updates on the coarsest layer, modifications are projected recursively down towards the finer layers and the simulation continues with the next time step.

7 EVALUATION

7.1 Productivity

Table 9 lists absolute values for code metrics collected for the AllScale and MPI versions of iPIC3D and AMDADOS, in order to get a grasp on the productivity of working with the AllScale API compared to MPI. *P.SLOC* denotes the lines of code spent on parallelizing the application, counting only the minimal set of lines

containing explicit interface calls. Additional code required for e.g. preparing arguments is not accounted for, and hence these results present a best-case perspective for MPI. Nevertheless, as the numbers show, AllScale clearly outperforms MPI. Furthermore, we have measured the sum of the cyclomatic complexity [15] (*TOT CY*) as well as the total count of non-comment lines of code (*SLOC*) over all translation units. Compared to *P.SLOC*, these give an indication of how much of the overall pilot application complexity is related to their manual MPI parallelization rather than actual domain science content.

	AMDADOS		iPIC3D	
	AllScale	MPI	AllScale	MPI
P.SLOC	25	70	23	56
SLOC	1 136	1 420	1 443	1 717
TOT CY	154	181	204	264

Table 9. Pilot application code metrics

Note that, in addition to greatly reducing the user-facing complexity of implementing distributed memory parallel programs, the AllScale versions inherently provide the possibility of integrating advanced features such as hybrid distributed/shared memory parallelism, inter-node load balancing, overlapping of communication and computation, or high-level monitoring facilities. All these features are not present in the MPI versions, and thus not accounted for in the comparisons presented here.

7.2 Performance

In order to ascertain the performance of the AllScale API and the underlying AllScale toolchain, we conducted weak scaling experiments for AMDADOS and iPIC3D on the Vienna Scientific Cluster (VSC-3) and the Meggie cluster of the University of Erlangen-Nuremberg. Table 10 lists their hardware characteristics. The initial problem size for a single node was chosen such that application throughput did not noticeably improve when further increasing the problem size.

Name	Nodes	CPU (Intel Xeon)	RAM	Interconnect	Compiler	MPI
VSC-3	512	2x E5-2650 v2	64 GB	IB QDR-80	GCC 7.2	OpenMPI 3.0.0
Meggie	256	2x E5-2630 v4		OPA 100 GBit	GCC 7.3	Intel MPI 2018.2

Table 10. Experimental platform description. The number of nodes refers to the maximum used in this work.

Figure 7 compares the performance results of the AllScale implementations against MPI reference implementations, with performance measured as application throughput.

For AMDADOS the AllScale variant achieves higher performance on both the VSC-3 and Meggie cluster. On both systems up to 160 % higher performance is

obtained, in particular for executions involving a larger number of nodes. In case of the 512 node run on VSC-3, the degrading scalability of the MPI reference implementation lead to exceeding the available time limit for our jobs. We defined this limit as 30 times the execution time of a single node run.

For iPIC3D, on the other hand, the MPI implementation demonstrates nearly-optimal scalability on both the VSC-3 and Meggie cluster. The throughput per node remains almost constant throughout the range of evaluated system setups. The AllScale version, however, shows varying performance characteristics. Generally, good performance is observed for a small number of nodes. However, while on Meggie throughput on larger scale systems remains comparable to the MPI reference version, on VSC-3 a considerably lower throughput is observed, which remains constant between 4 and 128 nodes. Beyond 128 nodes, performance degrades considerably again.

These results demonstrate the feasibility of using automatically managed user-defined data structures in large-scale high performance applications.

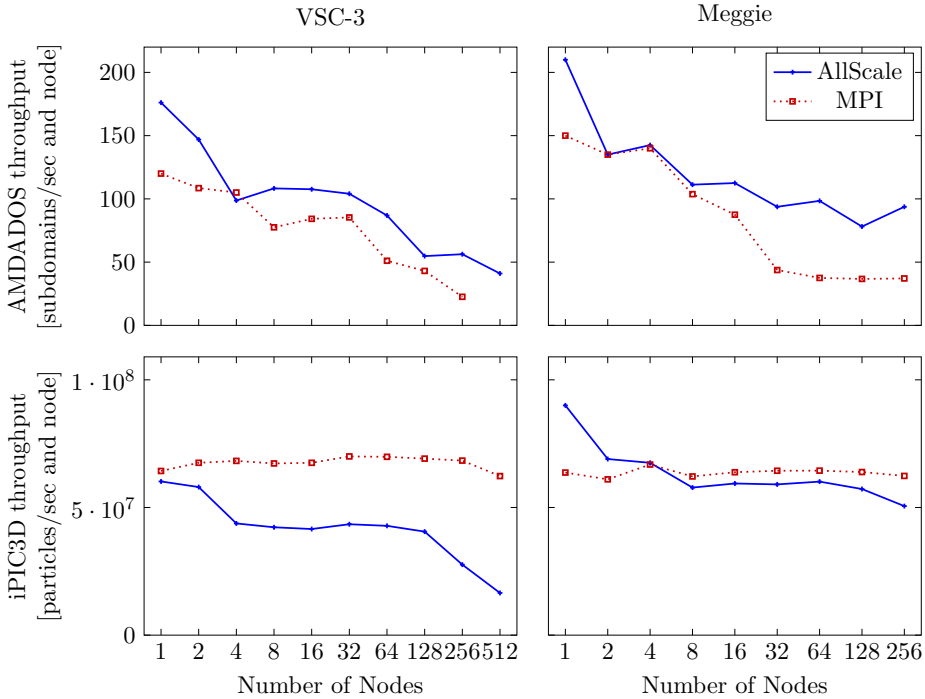


Figure 7. Comparison of throughput per node for AMDADOS and iPiC3D on VSC-3 and Meggie for MPI and AllScale

8 RELATED WORK

Conventional, low-level HPC infrastructures comprising combinations of MPI with some per-node parallelism APIs are still the default platforms for building HPC applications, but require programmers manually implement workload decomposition. Systems providing a higher level of abstraction, such as the AllScale API, can be grouped into three broad categories: new general purpose languages, domain-specific frameworks, and general purpose libraries. Note that there is a large number of parallelism approaches constrained to single-node shared memory hardware. We omit these from our overview provided here as they do not address the same problem space as the AllScale API.

In terms of languages, X10 [7] and Chapel [6] have targeted (recursive) parallelism on large scale, distributed systems, but left locality and data management to the user. Charm++ [13], on the other hand, is a C++ extension aiming at isolating the user from low-level mapping activities, thus facilitating portability. Its design is based on message-exchanging entities exposed to the user and lacks automated data distribution management. Recently, the ANTAREX research project [19] proposed a DSL-based approach, facilitating the separation of concerns between functional and non-functional aspects of HPC applications. However, due to its DSL-focused design, users require additional tools and may not rely on the experience of an established developer community.

Several new frameworks such as Lift [20], Delite [5], or AnyDSL [18] provide environments for implementing DSLs. Internally, DSL constructs are encoded using functional IR constructs like `map`, `reduce`, or `zip`. However, the resulting programming interface for the domain experts remains a DSL, targeting very specific application domains and inheriting the difficulties of DSLs noted above.

Domain-specific, C++-based libraries such as PETSc [4] or TensorFlow [1] handle several of the challenges addressed by our framework successfully for their respective domains. However, they are tailored towards specific domains instead of supporting a wider range of applications.

More general purpose parallel C++ library based frameworks like STAPL [3] and Kokkos [8] are exercising control over parallel algorithms and data structures similar to our architecture. STAPL envisions a separation of concerns strategy similar to ours. Kokkos, on the other hand, has a strong focus on multidimensional arrays and parallel loops, unlike the wider range of data structures and operations supported by our architecture. Due to a lack of compiler integration, these approaches require data dependencies of code regions to be expressed explicitly as part of the API, while this is covered implicitly in our approach.

9 CONCLUSION

This work presented the AllScale API, a novel interface for implementing distributed memory parallel applications with the programmability of a shared memory API. We illustrated how the distinction into the User and Core components provides

a separation of concerns for domain experts, HPC experts and system-level experts, and discussed several constructs of the AllScale API in detail. In addition, the three use cases presented show the suitability of our approach to real-world scientific problems, evaluated in both productiveness and parallel performance.

Future work includes better user feedback for programming errors, additional pre-provided operators in the User API along with new applications.

Acknowledgements

This project has received funding from the European Unions Horizon 2020 research and innovation programme as part of the FETHPC AllScale project under grant agreement No. 671603 and from the FFG (Austrian Research Promotion Agency) project INPACT, project No. 868018. The computational results presented have been achieved in part using the Vienna Scientific Cluster and the Regionales Rechen-Zentrum Erlangen.

REFERENCES

- [1] ABADI, M.—AGARWAL, A.—BARHAM, P.—BREVDO, E.—CHEN, Z.—CITRO, C.—CORRADO, G. S.—DAVIS, A.—DEAN, J.—DEVIN, M. et al.: Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2016, arXiv preprint arXiv:1603.04467.
- [2] AKHRIEV, A.—O'DONNCHA, F.—GSCHWANDTNER, P.—JORDAN, H.: A Localised Data Assimilation Framework within the 'AllScale' Parallel Development Environment. OCEANS 2018 MTS/IEEE Charleston, 2018, pp. 1–7, doi: 10.1109/OCEANS.2018.8604556.
- [3] AN, P.—JULA, A.—RUS, S.—SAUNDERS, S.—SMITH, T.—TANASE, G.—THOMAS, N.—AMATO, N.—RAUCHWERGER, L.: STAPL: An Adaptive, Generic Parallel C++ Library. International Workshop on Languages and Compilers for Parallel Computing, Springer, 2001, pp. 193–208, doi: 10.1007/3-540-35767-X_13.
- [4] BALAY, S.—ABHYANKAR, S.—ADAMS, M.—BRUNE, P.—BUSCHELMAN, K.—DALCIN, L.—GROPP, W.—SMITH, B.—KARPEYEV, D.—KAUSHIK, D. et al.: PETSc Users Manual Revision 3.7. Technical Report ANL-95/11 Rev 3.7, Argonne National Laboratory (ANL), Argonne, United States, 2016.
- [5] BROWN, K. J.—SUJEETH, A. K.—LEE, H. J.—ROMPF, T.—CHAFI, H.—ODERSKY, M.—OLUKOTUN, K.: A Heterogeneous Parallel Framework for Domain-Specific Languages. 2011 International Conference on Parallel Architectures and Compilation Techniques (PACT), IEEE, 2011, pp. 89–100, doi: 10.1109/PACT.2011.15.
- [6] CHAMBERLAIN, B. L.—CALLAHAN, D.—ZIMA, H. P.: Parallel Programmability and the Chapel Language. The International Journal of High Performance Computing Applications, Vol. 21, 2007, No. 3, pp. 291–312, doi: 10.1177/1094342007078442.

- [7] CHARLES, P.—GROTHOFF, C.—SARASWAT, V.—DONAWA, C.—KIELSTRA, A.—EBCIOGLU, K.—VON PRAUN, C.—SARKAR, V.: X10: An Object-Oriented Approach to Non-Uniform Cluster Computing. *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA '05)*, 2005, ACM SIGPLAN Notices, Vol. 40, 2005, No. 10, pp. 519–538, doi: 10.1145/1103845.1094852.
- [8] EDWARDS, H. C.—TROTT, C. R.—SUNDERLAND, D.: Kokkos: Enabling Many-core Performance Portability Through Polymorphic Memory Access Patterns. *Journal of Parallel and Distributed Computing*, Vol. 74, 2014, No. 12, pp. 3202–3216, doi: 10.1016/j.jpdc.2014.07.003.
- [9] JORDAN, H.—GSCHWANDTNER, P.—THOMAN, P.—ZANGERL, P.—HIRSCH, A.—FAHRINGER, T.—HELLER, T.—FEY, D.: The AllScale Framework Architecture. *Parallel Computing*, Vol. 99, 2020, Art.No. 102648, doi: 10.1016/j.parco.2020.102648.
- [10] JORDAN, H.—HELLER, T.—GSCHWANDTNER, P.—ZANGERL, P.—THOMAN, P.—FEY, D.—FAHRINGER, T.: The AllScale Runtime Application Model. *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, 2018, pp. 445–455, doi: 10.1109/CLUSTER.2018.00088.
- [11] JORDAN, H.—IAKYMCHUK, R.—FAHRINGER, T.—THOMAN, P.—HELLER, T. et al.: D2.4 – AllScale System Architecture (b). May 2018, [http://www.allscale.eu/docs/D2.4%20-%20AllScale%20System%20Architecture%20\(b\).pdf](http://www.allscale.eu/docs/D2.4%20-%20AllScale%20System%20Architecture%20(b).pdf).
- [12] JORDAN, H.—THOMAN, P.—ZANGERL, P.—HELLER, T.—FAHRINGER, T.: A Context-Aware Primitive for Nested Recursive Parallelism. In: Desprez, F. et al. (Eds.): *Euro-Par 2016: Parallel Processing Workshops*. Springer, Cham, *Lecture Notes in Computer Science*, Vol. 10104, 2017, pp. 149–161, doi: 10.1007/978-3-319-58943-5_12.
- [13] KALE, L. V.—KRISHNAN, S.: CHARM++: A Portable Concurrent Object Oriented System Based on C++. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '93)*, 1993, ACM SIGPLAN Notices, Vol. 28, 1993, No. 10, pp. 91–108, doi: 10.1145/167962.165874.
- [14] MARKIDIS, S.—LAPENTA, G.—RIZWAN-UDDIN: Multi-Scale Simulations of Plasma with iPIC3D. *Mathematics and Computers in Simulation*, Vol. 80, 2010, No. 7, pp. 1509–1519, doi: 10.1016/j.matcom.2009.08.038.
- [15] MCCABE, T. J.: A Complexity Measure. *IEEE Transactions on Software Engineering*, Vol. SE-2, 1976, No. 4, pp. 308–320, doi: 10.1109/TSE.1976.233837.
- [16] NUMECA International. 2019, <https://www.numeca.com/>.
- [17] O'DONNCHA, F.—IAKYMCHUK, R.—AKHRIEV, A.—GSCHWANDTNER, P.—THOMAN, P.—HELLER, T.—AGUILAR, X.—DICHEV, K.—GILLAN, C.—MARKIDIS, S.—LAURE, E.—RAGNOLI, E.—VASSILIADIS, V.—JOHNSTON, M.—JORDAN, H.—FAHRINGER, T.: AllScale Toolchain Pilot Applications: PDE Based Solvers Using a Parallel Development Environment. *Computer Physics Communications*, Vol. 251, 2020, Art.No. 107089, doi: 10.1016/j.cpc.2019.107089.
- [18] LEISSA, R.—BOESCHE, K.—HACK, S.—PÉRARD-GAYOT, A.—MEMBARTH, R.—SLUSALLEK, P.—MÜLLER, A.—SCHMIDT, B.: AnyDSL: A Partial Evaluation Framework for Programming High-Performance Libraries. *Proceedings of the ACM*

- on Programming Languages, Vol. 2, 2018, Issue OOPSLA, Art.No. 119, doi: 10.1145/3276489.
- [19] SILVANO, C.—AGOSTA, G.—CHERUBIN, S.—GADIOLI, D.—PALERMO, G.—BARTOLINI, A.—BENINI, L.—MARTINOVIČ, J.—PALKOVIČ, M.—SLANINOVÁ, K.—BISPO, J. A.—CARDOSO, J. M. P.—ABREU, R.—PINTO, P.—CAVAZZONI, C.—SANNA, N.—BECCARI, A. R.—CMAR, R.—ROHOU, E.: The ANTAREX Approach to Autotuning and Adaptivity for Energy Efficient HPC Systems. Proceedings of the ACM International Conference on Computing Frontiers (CF'16), ACM, 2016, pp. 288–293, doi: 10.1145/2903150.2903470.
 - [20] STEUWER, M.—REMMELG, T.—DUBACH, C.: LIFT: A Functional Data-Parallel IR for High-Performance GPU Code Generation. 2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), IEEE, 2017, pp. 74–85, doi: 10.1109/CGO.2017.7863730.
 - [21] THORSON, G.—WOODACRE, M.: SGI UV2: A Fused Computation and Data Analysis Machine. Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12), IEEE, 2012, Art.No. 105, doi: 10.1109/SC.2012.102.



Philipp Gschwandtner is a senior scientist at the Research Center HPC at the University of Innsbruck, Austria. He completed his Ph.D. in 2017 with a thesis on performance and energy analysis and optimization of parallel programs. His main research interests lie in high performance computing and parallel programming, including adjacent topics such as scientific computing, program optimization, API design, runtime systems, and compiler research.



Herbert Jordan is a former PostDoc at the University of Innsbruck, Austria. After completing his Ph.D. in 2014 on the Insieme optimizing compiler infrastructure he became the Scientific Coordinator of the EU H2020 project AllScale, aimed at exposing nested recursive parallelism for distributed memory. His main research interests are in programming language and API design, static program analysis, code transformations, and performance optimizations.



Peter THOMAN is Assistant Professor of computer science at the University of Innsbruck. His research focus has been to improve the effective performance and programmability of complex, highly parallel systems. He has created and contributed to various APIs for HPC and GPGPU platforms, such as the Celerity high-level single-source platform for GPU Clusters. He also investigates runtime systems, as well as compiler-based tools for performance optimization as well as improved developer support.



Thomas FAHRINGER is Professor of computer science at the University of Innsbruck since 2003. His research focuses on supporting researchers in science and engineering by developing, analysing, and optimizing parallel and distributed applications. Fahringer was involved in numerous national and international research projects including 15 EU funded projects, published 5 books, 40 journal and magazine articles, and more than 200 reviewed conference papers. He is a member of the IEEE and ACM.

PROCESSING RADIO ASTRONOMICAL DATA USING THE PROCESS SOFTWARE ECOSYSTEM

Souley MADOUGOU, Hanno SPREEUW, Jason MAASSEN

Netherlands eScience Center

Science Park 140 (Matrix I)

1098 XG Amsterdam, The Netherlands

e-mail: {s.madougou, h.spreeuw, j.maassen}@esciencecenter.nl

Abstract. In this paper we discuss our efforts in “unlocking” the Long Term Archive (LTA) of the LOFAR radio telescope using the software ecosystem developed in the PROCESS project. The LTA is a large (> 50 PB) archive that expands with about 7 PB per year by the ingestion of new observations. It consists of coarsely calibrated “visibilities”, i.e. correlations between signals from LOFAR stations. Converting these observations into sky maps (images), which are needed for astronomy research, can be challenging due to the data sizes of the observations and the complexity and compute requirements of the software involved. Using the PROCESS software environment and testbed, we enable a simple point-and-click-reduction of LOFAR observations into sky maps for users of this archive. This work was performed as part of the PROCESS project which aims to provide generalizable open source solutions for user friendly exascale data processing.

Keywords: Radio astronomy, imaging, extreme large scale data processing, PC clusters, distributed computing, grid, cloud computing

Mathematics Subject Classification 2010: 68-04

1 INTRODUCTION

The LOw Frequency ARray (LOFAR) [40] is a European radio telescope which covers frequencies between 10 and 250 MHz. Designed by ASTRON [18], it became operational in 2010, and its design differs from classical radio telescopes that usually consist of arrays of dishes. Instead, LOFAR combines the signals from a large

number of relatively simple omnidirectional antennas, shown in Figure 1. These antennas are grouped into stations, each typically consisting of 96 Low Band Antennas (10 MHz–90 MHz) and 48 High Band Antennas (110 MHz–250 MHz).

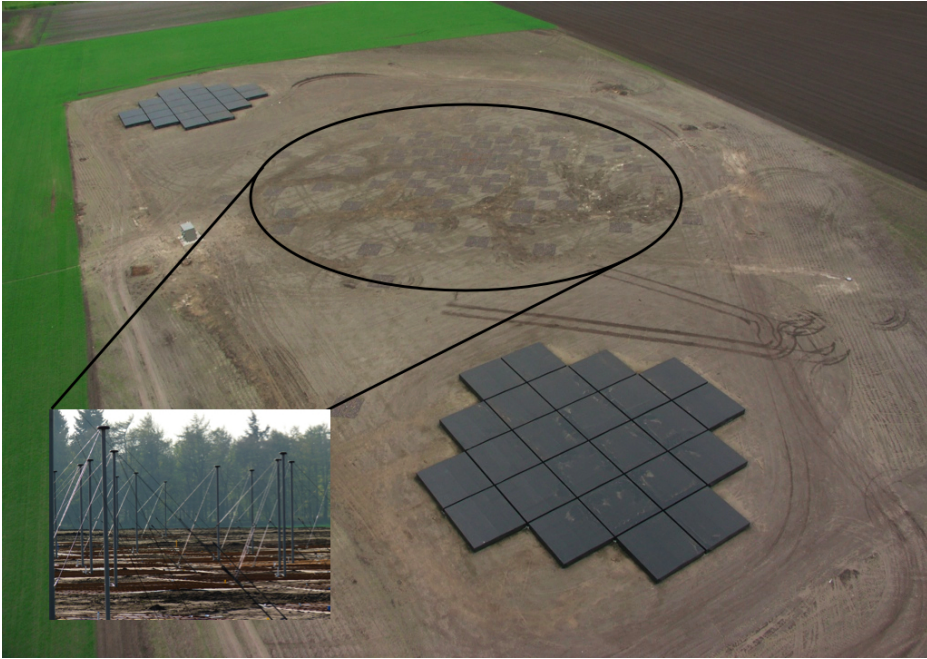


Figure 1. LOFAR LBA (poles) and HBA (boxes) antenna types forming a single station (image courtesy of ASTRON)

The signals received by these antennas are combined into a single station signal, similar to the signal of a single dish from a classical radio telescope. LOFAR currently consists of 52 stations in total (as shown in Figure 2): 24 core stations located within a 2 km radius near the village of Exloo in the east of the Netherlands, 14 additional stations in the Netherlands arranged in an (approximated) logarithmic spiral distribution, and 14 international stations located in Germany, France, Sweden, UK, Poland, Ireland and Latvia and (in future) Italy.

For observations the station signals are correlated per pair of two stations, called a *baseline*, following the principles of aperture synthesis. Every pair of signals, each consisting of a sequence of complex numbers, is multiplied with each other and a complex phase (which determines the direction of the observation), and integrated over the time sampling interval. Depending on the combination of stations used, LOFAR supports baselines from a few hundred meters to several thousand kilometers. While longer baselines provide a higher angular resolution, they complicate signal calibration as ionospheric disturbances vary more over longer distances. Every LOFAR imaging observation results in a large set of such correlations, called



Figure 2. LOFAR station distribution over Europe, including planned LOFAR stations (image courtesy of ASTRON)

visibilities. A visibility is recorded as a complex number for each baseline, frequency and time sampling interval, and polarization product.

The LOFAR Long Term Archive [36] (LTA) was set up to store all LOFAR observations. A typical LOFAR observation takes 8–12 hours and has a size of about 100 TB. Frequency averaging of every eight channels reduces this size to about 16 TB. Initial (coarse) calibration is also applied to improve the signal quality. The last few years the LTA has been expanding by about 7 PB per year and is currently (2020) exceeding 50 PB. The LOFAR LTA is stored on tapes at locations in Amsterdam (The Netherlands), Jülich (Germany) and Poznan (Poland).

LOFAR science drivers are condensed in six key science projects [13] (KSPs): the epoch of reionisation, deep extragalactic surveys, transient sources, ultra-high energy cosmic rays, solar science and space weather, and cosmic magnetism. In addition, the LOFAR data is publicly available for other uses.

The coarsely calibrated observations stored in the LTA are not directly suitable as a starting point for scientific research. KSP-specific processing pipelines are needed to further refine the observations into science products, using different combinations of processing tools and parameters. Each processing step is complex and usually requires both domain and software knowledge to generate useful output. Combined with the massive volumes of the data, further processing of the data

within the LTA is hard for non-experts. These challenges are exacerbated further when one needs to process not just one, but many observations.

In this paper, we describe our efforts to build a user-friendly “point-and-click-processing” system for the users of LTA data: after selecting an observation, an appropriate pipeline and (optionally) a set of parameters, acquiring a well-calibrated sky map just requires waiting for the processing to complete. Staging the data (copying from magnetic tape to disk), transferring this data to a suitable compute infrastructure, launching the processing pipeline, and retrieving the results, is all handled automatically by the platform. We focus on the pipeline producing sky maps, as these typically serve as a starting point for astronomy research. However, we believe the approach is generalizable to other pipelines used for processing LTA data.

This research was conducted as one the five use cases of the EU H2020 *PROCESS* project [21]. In Europe, we currently do not have any exascale supercomputers. Therefore, any form of processing requiring exascale data processing will have to be distributed over a number of clusters in Europe. Such distributions over many clusters will have to be performed seamlessly and all software packages that run the computations will have to be containerized to guarantee portability. The goal of *PROCESS* is to offer exascale computing service prototypes to a range of scientists that require big storage and big compute facilities, in such a way that these users can remain mostly agnostic of the location and specifications of the compute clusters where their data will be processed. Portability and scalability are the main requirements for the *PROCESS* software infrastructure, not only with respect to compute, but also with respect to data access.

2 RELATED WORK

One of the KSPs of the LOFAR telescope is to conduct deep wide-field surveys. For instance, the LOFAR Two Meter Sky Survey (LoTSS) [38] is observing 3 000 different fields that will collectively map the entire northern radio sky and create a total of 48 Petabytes of raw observation data that will be stored in LTA. Typically each dataset is 16 TB and is split into 244 files of 65 GB. This data is further processed through the LOFAR imaging pipeline. To complete the LoTSS survey in the project’s target five year duration, multiple datasets need to be processed on a daily basis. To cope with the challenge, the LoTSS community has built the LRT (LOFAR Reduction Tools) framework [33] that provides automation, portability, scalability and generalisation. LRT is built on top of a work distribution environment which dispatches LOFAR pre-processing on a computing cluster [35] using a PiCaS server [19] to track progress. Both because of this legacy and the required high transfer rates, the platform has to run on a large computing infrastructure connected to the LTA with high-speed network, which limits its use to that infrastructure. Furthermore, the parallelisation only concerns a single step in each of the calibrator and the target pipelines.

In an extension of the work described above, the same authors present AGLOW or Automated Grid-enabled LOFAR Workflows [34]. AGLOW is a workflow orchestration system that integrates LOFAR processing with a distributed computing platform. It uses Dutch Grid infrastructure and is based on Apache Airflow [2]. According to the authors, AGLOW allows to reduce the setup of complex workflows from months to days. Both contributions are focused on lowering the data reduction time by means of distributed computing. While this was a desirable feature for our use case, our focus is more concerned with ease of use and portability of existing pipelines.

Other authors [37] have investigated the viability of the cloud as infrastructure for processing LOFAR calibration pipeline as opposed to the more traditional dedicated clusters and the grid. They found that while the cloud presents some advantages such as the ease of software installation and maintenance and the automatic scale-out, the most interesting ones, the commercial platforms, are also more expensive than the use of a dedicated cluster for large datasets. The cloud solution is only competitive if the number of datasets to be analysed is not high, which disqualifies it, for instance, for surveys KSP. Furthermore, the pipeline tools used for the tests do not include the most recent developments such as *FACTOR* or DDF.

3 BACKGROUND

This section consists of two parts where we lay down the foundations for understanding the science case behind our use case and the environment in which it is implemented. First, we describe the LOFAR imaging pipeline with enough details to understand the choices made for our use case implementation in PROCESS. Next, we briefly summarise the PROCESS project and describe its ecosystem components and architecture.

3.1 The LOFAR Imaging Pipeline

Astronomers often embark on a scientific investigation by inspecting sky maps. For example, after the detection of a cosmic explosion by a gamma-ray satellite at a particular position on the sky, an astronomer will want to find out which celestial objects are visible near this position on the sky at other wavelengths, like radio. This is where a repository with low frequency radio maps covering a large part of the sky can provide a useful resource. Unfortunately, such sky maps are currently only available for a fraction of the LOFAR observations stored in the LTA (as produced by the LOFAR Two Meter Sky Survey (LoTSS) [38] for example).

When an astronomer wants to produce new sky maps, data first needs to be downloaded from the LTA. ASTRON provides a convenient web portal, shown in Figure 3, which allows users to search through the available data and select the observations which need to be retrieved from tape. Once the data is retrieved, it must be downloaded from temporary disk storage at the LTA to the users own infrastructure. The size of these datasets can be significant, up to 16 TB per observation.

LOFAR

Long Term Archive

HOME

SEARCH DATA

BROWSE PROJECTS

HELP

LOGIN

Observation 1 to 100 (showing 100 of total 41189) -

edit columns

first previous 1 2 3 4 5 6 7 8 9 10 11 ... next last

#	Project	Release Date	SAS Id	Antenna Set	Instrument Filter	Channel Width [MHz]	Number Of SubArray Pointings	Nr Stations Core	Nr Stations Remote	Nr Stations International	Number Of Stations	Number Of Correlated DataProducts	Number Of BeamFormed DataProducts
100	LT14_004	793198	HBA Dual Inner	110-190 MHz	0.000000	3	24	14	11	49	486 / 487	0	
99	LT14_004	793200	HBA Dual Inner	110-190 MHz	0.000000	1	24	14	11	49	243	0	
98	LT14_002	792948	LBA Outer	30-90 MHz	0.000000	4	24	13	0	37	488	0	
97	LT14_002	792958	LBA Outer	30-90 MHz	0.000000	4	24	13	0	37	488	0	
96	LT14_002	792900	LBA Outer	30-90 MHz	0.000000	4	24	13	0	37	488	0	
95	LT14_002	791294	LBA Outer	30-90 MHz	0.000000	4	24	14	0	38	488	0	
94	LT14_002	791304	LBA Outer	30-90 MHz	0.000000	4	24	14	0	38	488	0	
93	LT14_002	791314	LBA Outer	30-90 MHz	0.000000	4	24	14	0	38	488	0	
92	LC14_003	2021-09-03	793556	HBA Dual	110-190 MHz	0.000000	1	24	0	0	24	244	4
91	LC14_003	2021-09-03	793560	HBA Dual	110-190 MHz	0.000000	1	24	0	0	24	244	8
90	LC14_003	2021-09-03	793564	HBA Dual	110-190 MHz	0.000000	1	24	0	0	24	244	4
89	LT14_002	791324	LBA Outer	30-90 MHz	0.000000	4	23	14	0	37	488	0	
88	LT14_002	791334	LBA Outer	30-90 MHz	0.000000	4	23	14	0	37	488	0	
87	LT14_002	791466	LBA Outer	30-90 MHz	0.000000	4	23	14	0	37	488	0	
86	LT14_002	791476	LBA Outer	30-90 MHz	0.000000	4	23	14	0	37	488	0	
85	LT14_002	791344	LBA Outer	30-90 MHz	0.000000	4	23	14	0	37	488	0	
84	LT14_004	793210	HBA Dual Inner	110-190 MHz	0.000000	1	24	14	11	49	243	0	
83	LT14_004	793212	HBA Dual Inner	110-190 MHz	0.000000	3	24	14	11	49	486 / 487	0	
82	LT14_004	793214	HBA Dual Inner	110-190 MHz	0.000000	1	24	14	11	49	243	0	

Figure 3. The LTA web interface for searching and downloading observation data, accessible through <https://lta.lofar.eu/Lofar> (image courtesy of ASTRON)

To produce images from this observation data, a large number of processing steps need to be performed, as shown in Figure 4 (in a simplified form). This pipeline is generally referred to as the Standard Imaging Pipeline (SIP) [16].

The initial steps, known as the Default Pre-Processing Pipeline (DPPP) [5], constitute (among other things) of *flagging* the data to remove radio frequency interference (RFI), optionally *averaging* to reduce the data volume, and *demixing* to subtract the contributions of the brightest sources in the sky to increase the sensitivity.

Next, an initial set of calibration parameters is applied. To do so, a short observation of a reference source (the *calibrator*) is performed immediately preceding or succeeding the main observation of the *target* (the astronomical source of interest). A Local Sky Model (LSM) which matches the area of interest is then extracted from the LOFAR Global Sky Model (GSM). The GSM is a “ground truth” database containing all known sources from various sky survey catalogs, including VLSS [28], WENSS [30], TGSS [31]. Using the LSM and calibrator observation in an iterative process, an estimate can be obtained for instrumental and environmental effects such as electronic station gains and ionospheric delays. The target observation can then be corrected for these effects, a step generally referred to a Direction Independent calibration (DI).

The calibrated data are then converted into an image using an imager that applies the w-projection algorithm [29] to remove the effects of noncoplanar baselines when imaging large fields and the A-projection algorithm [39] to take into account the varying primary beam during synthesis observations. The LSM is expanded and updated in the process by extracting sources from the images. One or more loops

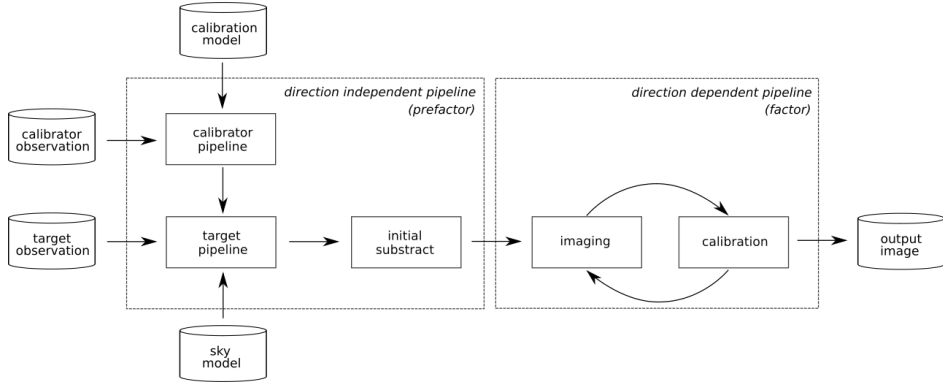


Figure 4. Imaging pipeline main steps

of calibration, imaging and LSM updates are performed. At the end of the process, the final LSM will be used to update the GSM, and the final images are generated. Various types of calibration algorithms can be used in this process, depending on the requirements and user preferences.

Creating such an imaging pipeline is complex and requires detailed knowledge of the domain, tools and pipeline framework (not to mention significant programming skills). Therefore, a *genericpipeline* [9] is offered by ASTRON, which helps astronomer design their processing pipeline without requiring too much technical knowledge. This *genericpipeline* contains predefined pipeline steps for the user to choose from. Creating a new pipeline then boils down to defining a so-called *parset*; a parameter set (or pipeline definition) which selects and configures the relevant steps in the *genericpipeline*.

A tool commonly used in this process is *prefactor* [20] which consists of various *parsets* for the *genericpipeline* to steer the processing of LOFAR data. Originally intended to prepare the data for input to the direction-dependent (DD) calibration software *FACTOR* [7] (hence, its name), *prefactor* performs the steps described above to correct for various instrumental and ionospheric effects on observations, and makes the observation ready for more advanced DD calibration pipelines, such as *FACTOR* or *killMS*.

In this paper, we decided to use *FACTOR*, as we found it to be one of the most stable tools available for DD calibration. It produces low-noise, high-resolution images from HBA LOFAR data using the facet calibration scheme [41]. *FACTOR* corrects for direction-dependent effects, including ionospheric effects and beam-model errors. *FACTOR* works by dividing up the target observation field into many facets and separately solving for the direction-dependent corrections in each facet. It is designed to minimize the number of free parameters needed to parameterize these corrections to avoid overfitting. This minimization is critical in producing high-fidelity images.

While the available tools are generally well documented and several LOFAR imaging tutorials can be found online, the overall process is quite cumbersome for non-expert users. A large number of tools must be installed and configured, sometimes resulting in complex technical or software dependency problems. Once the tools are installed successfully and configured correctly, the data volumes that need to be transferred and processed are significant and often exceed the capabilities of the infrastructure available to the users. Therefore, a user-friendly “point-and-click-processing” system for the users of LTA data could significantly lower the threshold for using LOFAR LTA data for the non-expert users. In Section 4, we describe how we have implemented such a system as one the five use cases using the software environment developed in the *PROCESS* project, which is briefly described below.

3.2 *PROCESS*

The aim of the *PROCESS* project [21] is to provide an open-source, multi-purpose and scalable software environment specially developed for exascale data processing. This goal was achieved by creating various tools and services that support set of heterogeneous extreme scale data processing use-cases driven by both the scientific research community and industry [22].

Although these use cases come from very different communities (medical imaging, radio astronomy, airline ancillary pricing, disaster risk management and earth observation), they share the same problems:

1. they need to process very large volumes of data using a diverse collection of tools,
2. these tools are difficult to install and configure by the users who often lack the necessary technical knowledge, and
3. the storage and/or compute requirements exceed the capabilities of the infrastructure that is available to the users.

During the course of the *PROCESS* project, a modular service architecture was designed and implemented [26], a simplified schematic of which is shown in Figure 5. It can be divided into three main modules: the Interactive Execution Environment [10] (IEE), which provides a web-based user API (as well as a REST API) for submitting pipelines, the LOBCDER data module [12], which offers distributed storage and data transfer services, and the compute module which provides access to both HPC and Cloud compute infrastructure (via RimRock [23] and Cloudify [3], respectively). For testing purposes, these services are deployed on compute and storage infrastructure at Cyfronet¹ and LRZ².

¹ <http://www.cyfronet.krakow.pl/en>

² <https://www.lrz.de/english>

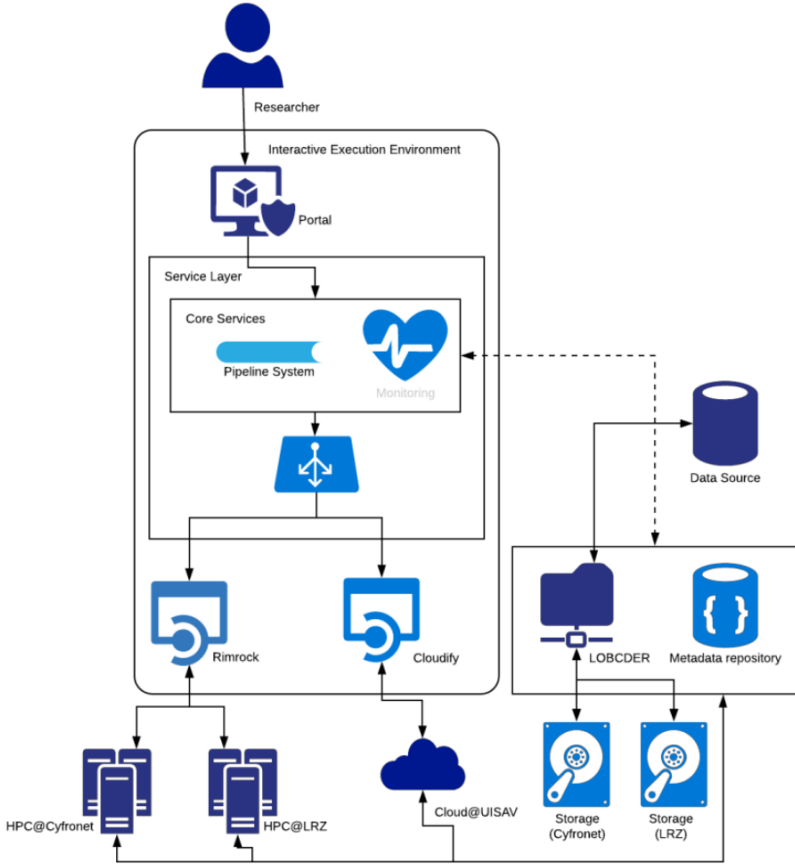


Figure 5. PROCESS platform architecture (image courtesy of [27])

To run a processing pipeline, the user will need a containerized version of the necessary tools. Currently, only Singularity [32] containers are supported. Creating such a containerized version of the tools may be too complex for the average user. However, there is a current trend among tool developers to provide such containers themselves, as this is seen as an easy solution to the dependency problems often encountered by users when installing software.

Using the IEE, a processing pipeline can be defined based on the containerized tools and then submitted to the compute infrastructure (either based on HPC or Cloud technology). Before the processing starts, the IEE will use the LOBCDER data module to transfer the necessary input data from the source location to the selected compute infrastructure.

The architecture designed by PROCESS provided us with the necessary services and infrastructure to create a “point-and-click” solution for the LTA imaging pipeline, which will be described in the next section.

4 POINT-AND-CLICK PROCESSING IMPLEMENTATION

To create an easy to use solution for the LTA imaging pipeline, the requirement analysis showed that three components need to be in place:

- A use-case specific web user interface, that enables users to select the desired datasets and processing pipelines.
- Containerized versions of the LOFAR imaging tools.
- Data services for retrieving the observation data from tapes at the LTA (staging), transferring this data to compute infrastructure, and the extraction of resulting images.
- Compute services to run the containerized LOFAR imaging tools on the compute infrastructure.

The solution we have implemented using the PROCESS services is shown in Figure 6, and fulfills all of these requirements. We will first provide an overview of how these components interact and then describe each of the components in more detail below.

At the startup of the web application, the backend connects to the database at the LTA archive and extracts a list of all accessible observations (*step 1*). Next, it retrieves the list of pre-configured pipelines available to the user from local storage (*step 2*). Both are then presented to the user in the web frontend, which is described in more detail in Section 4.1.

Once the user has selected a suitable target and calibrator observation (*step 3*), all necessary information on the selected observations, the pipeline, and various parameters are submitted to the IEE (*step 4*, described in Section 4.2). Before the IEE can execute the pipeline on the compute infrastructure, however, the observation data must be retrieved from the LTA archive. To do so, the IEE requests that the LOBCDER data service to retrieve the necessary data from the LTA (*steps 5-8*, explained in Section 4.3).

Once the data is available, the IEE will execute the pipeline on the compute infrastructure (*steps 9 and 10*) using a containerised version of the pipeline (Section 4.4). Finally, the result is returned to the the user via the web frontend (*steps 11 and 12*, Section 5).

4.1 Web Frontend

To enable easy access to the data processing infrastructure, we decided to create a use-case specific web portal, based on LTACAT [14], which was developed earlier

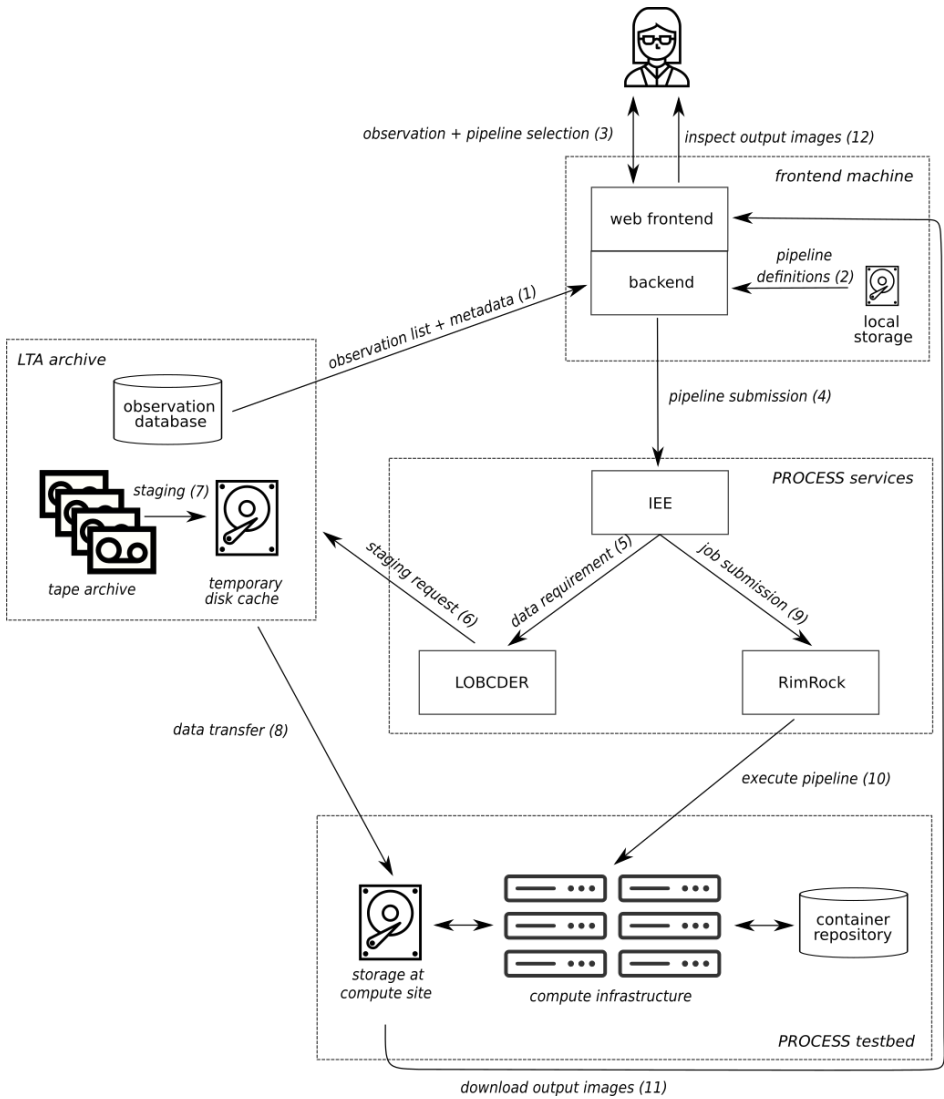



Figure 6. Schematic workflow of interaction between the images processing web application, PROCESS services, and the LTA archive

in the EOSC pilot for LOFAR project [6]. The latter is a React³ application based on FRBCAT [8] (originally developed as a catalogue for fast radio bursts (FRBs) in the AA-ALERT [1] project). This web application was customised and extended to fit LOFAR LTA imaging pipeline needs, and can be found at [15].



UC#2: SKA/LOFAR

The goal of this use case is to simplify the processing of archived data. Astronomers should be able to select a dataset on a portal, select a workflow, and then launch the processing pipeline from there. For this we need an easy to use, flexible, efficient and scalable workflow infrastructure for processing of extremely large volumes of astronomical observation data.

Through this portal, the astronomer must be able to browse through the available datasets and available workflows, and launch processing directly from there to the hardware infrastructure available in the project. Data should then be transferred from the LTA to the processing infrastructure, processed, and the results made available in the portal.

Choose Calibrator

OBSERVATIONID	STARTTIME	ENDTIME	RIGHTASCENSION	DECLINATION	NR_SUBBANDS
96651	2013-02-21T20:00:00.000Z	2013-02-22T00:59:59.000Z	56.70208333333333	68.09638888888889	244
215597	2014-04-06T06:14:00.000Z	2014-04-06T06:29:01.000Z	233.738625	23.5029166667	244
230427	2014-06-02T04:09:12.000Z	2014-06-02T04:19:11.000Z	212.835416667	52.2027777778	244
230499	2014-06-05T01:09:00.000Z	2014-06-05T01:26:59.000Z	233.738541667	23.5031388889	244
233982	2014-07-14T05:30:58.000Z	2014-07-14T05:30:58.000Z	289.452916667	6.35611111111	244

5

1 2 3 4 5 > >>

Choose Target

OBSERVATIONID	STARTTIME	ENDTIME	RIGHTASCENSION	DECLINATION	NR_SUBBANDS
96651	2013-02-21T20:00:00.000Z	2013-02-22T00:59:59.000Z	56.70208333333333	68.09638888888889	244
215597	2014-04-06T06:14:00.000Z	2014-04-06T06:29:01.000Z	233.738625	23.5029166667	244
230427	2014-06-02T04:09:12.000Z	2014-06-02T04:19:11.000Z	212.835416667	52.2027777778	244
230499	2014-06-05T01:09:00.000Z	2014-06-05T01:26:59.000Z	233.738541667	23.5031388889	244

Figure 7. Main LTA database view

The LTA database main view, shown in Figure 7, shows information and meta-data about LOFAR observations, similar to the original LTA web portal. As shown in Figure 6, the backend directly connects to the LTA database allowing users to seamlessly access the observation data archived in the LTA. Access to some observations may be restricted, however, due to various policies. Instead of providing a unified view of all data (as the original LTA web portal offers), the user is presented with two lists in order to select the calibrator and the target observations separately. The metadata can be used to filter the selection.

³ <https://reactjs.org>

Select processing pipeline:
UC2 IEE pipeline

Configuration Parameters: ☐

This is PROCESS UC2 pipeline for LOFAR observation calibration and imaging

staging ☐

Staging service URL
http://145.100.130.145:32010

User login on LTA
souley

User pass on LTA
<USER_LTA_PWD>

hpc ☐

search

HPC head node
pro.cyfronet.pl

Where observation will be transferred to on HPC
/net/archive/groups/plggprocess/UC2/input

Download service URL
http://lobcder.process-project.eu:8090

SRM certificate for transfer
<SRM_CERTIFICATE>

User login on HPC
plgsouley

User pass on HPC
<USER_HPC_PWD>

IEE Web service URL
https://process-dev.cyfronet.pl/api/projects/UC2/pipelines/lofar_pipeline/computations

IEE security token (JWT)
<IEE_JWT_TOKEN>

FACTOR working directory
/mnt/out/factor

Working directory
/mnt/workdir/test

Directory where transferred observations will be stored
/mnt/in

Name of the container image to be run
factor-iee.sif.old

PROCESS computing site name
Cyfronet: Prometheus

Submit workflow

Figure 8. Pipeline configuration and submission

Once the user selects a calibrator and target observations, the web application provides a separate window to select (and optionally configure) a processing pipeline, as shown in Figure 8. This window provides a list of available pipelines, the configuration parameters for the selected pipeline, and a submit button which will submit the pipeline execution to the IEE.

Currently, the definitions of these pipelines are stored locally on the machine running the Web application. In the backend, the pipeline configurator is automatically generated from a *JSON schema* [11] describing each pipeline. This allows for

predefined default values, constraints on inputs, mandatory required properties, and defining dependences between properties.

Each definition already includes suitable defaults for the pipeline parameters, such as configuration settings for the processing steps and the necessary configuration for the data retrieval and the choice of a computing site. Once set correctly, the users does not need to adjust these. Suitable defaults are provided for the current *PROCESS* testbed, which will retrieve data from the LTA and use the HPC facilities at Cyfronet as the compute site. If needed, these values can be adjusted in the *staging* and *hpc* sections of the pipeline configuration shown in Figure 8.

As part of the *PROCESS* testbed, the web portal comes bundled with a single predefined pipeline, but it is easily extended. It allows users to develop additional pipelines and integrate them in the Web application. For this purpose a pipeline template is provided in addition to a step-by-step guide on how to integrate new pipelines into the service [17]. The procedure consists of implementing a *run* function, defining the pipeline configuration parameters in *JSON schema* format, and registering the pipeline in the pipeline administrator of the IEE. After installation of the new pipeline based on these steps, a new pipeline appears in the list of available pipelines.

4.2 Pipeline Submission to the IEE

Once the pipeline is selected (and optionally configured), the user can submit it to the IEE for execution. The integration between the web portal and the IEE consists of REST API calls for retrieving the pipeline configuration parameters and submitting pipeline computations to the IEE based on the expected parameters.

When a pipeline is submitted, the LTA identifiers of the required target and calibrator observations are provided to the IEE as parameters of the pipeline. Before actual the actual pipeline execution begins, the IEE requests LOBCDER to fulfill the data requirements of the pipeline and ensure the data of both observations are available on the compute site. LOBCDER is described in more detail in the next section. This step may be skipped if the data already resides on the compute site.

Once the data is available, the pipeline job will be scheduled on the compute infrastructure. This job consists of a containerized version of the pipeline (described in Section 4.4 to ensure portability. This job submission is performed through Rim-Rock, which provides a REST API to the underlying scheduling system of the HPC compute cluster.

Once running, the web portal will retrieve the pipeline status from the IEE using the REST API calls. Once the pipeline has completed, the IEE will provide a link to where the results can be retrieved.

4.3 LOBCDER Data Services

Before the pipeline can be executed, the observation data needs to be available on the compute site. Consequently, the very first action of the IEE when receiving

a pipeline submission is to call the LOBCDER data services to request that the target and calibration observations are retrieved from the LTA archive. LOBCDER in turn contacts the LTA to request that the observational data is retrieved from the tapes and stored at a temporary location (a process referred to as “staging”). Once this staging is complete, LOBCDER will transfer the data to the selected compute site.

Therefore, to satisfy the data service requirements of our use case, several endpoints have been created by the LOBCDER team. One for issuing a staging command and one to check its status. Additionally, another one to transfer the staged in data from the temporary location to a HPC cluster for processing, along with its corresponding status check command. These are incorporated into the IEE portal using a Python module. The parameters for configuring these service endpoints are exposed by the JSON schema for the pipeline configuration.

Once the transfer is completed, LOBCDER notifies IEE which then submits the job which will run the pipeline to the workload management system on the selected computing resource(s).

4.4 Containerized Analysis Pipeline

To ensure portability, we have created a containerized version of the LTA imaging pipeline based on CWL [25] and Singularity⁴. This container essentially implements the *FACTOR* pipeline shown in Figure 4. The first two steps (*calibrator* and *target*) are taken care of by *prefactor* which provides parsets for each of them. They provide direction independent calibration. *prefactor* also provides a parset for the third step, *subtract*, which is specific to *FACTOR*. This step images the field at medium and low resolution to make initial models of the sources and subtracts these models from the *uv* data. The last step is *FACTOR* itself performing direction dependent calibration and imaging of HBA data. It divides the field into facets based on bright direction-dependent calibrators. It then cycles over the facets to self calibrate the calibrator sources (*facetselfcal*) and to improve the subtraction with new model and calibration (*facetsub*). The facets are then imaged (*facetimage*). Finally, *FACTOR* makes a mosaic of all facets and corrects for the primary beam attenuation (*field-mosaic*).

For the integration of the container with IEE, we need to provide an appropriate *run script* within the container. This script passes the expected input parameter values provided to the container by the IEE to the workflow runner (*cwl-runner*) running inside the container. These parameters are provided by the user via the frontend shown in Figure 8. They are passed to IEE through the REST API calls described above. Currently, the container is stored online at a private location, from where it is currently manually downloaded and installed at the computing site by the IEE team. We aim to automate this process by storing the container in a registry so it can be automatically retrieved, as shown in Figure 6.

⁴ <https://www.sylabs.io>

Once the processing pipeline has completed, the output is made available for download by the IEE via a download link. This output consists of the output images in FITS format⁵, which is commonly used in astronomy, plus inspection plots (examined by the astronomers to check the normal functioning of various parts of the instrument) and various log files produced by the processing steps in the pipeline. The full resolution output images can be downloaded by the user. For convenience, downscaled JPG thumbnails of the images are presented in the frontend.

5 RESULTS

With our use case fully implemented using the *PROCESS* services, generating images from the LTA data has never been so easy: the astronomer has just to launch the frontend, choose its calibrator, target and pipeline, and click a button. The *PROCESS* platform takes care of all the processing. The user can then use his/her valuable time for more analytical tasks while waiting for the images. Sample output images generated by this pipeline are shown in Figure 9.

Figure 9a) shows the image that will be produced when directly using the data from the LTA archive. This data is only roughly calibrated, and contains significant distortions caused by ionospheric interferences and instrument effects. Figure 9b) shows the data after the initial processing, demixing of bright sources, and direction independent calibration, which already improves the image quality significantly. Figure 9c) shows the final result after direction dependent calibration, which further improves the image quality to the level required for astronomy research.

Step	Data Size [GB]	Run Time [s]
calibrator	25	8 534
target	433	11 902
subtract	76	37 212
FACTOR	76	464 400 (~ 5 d9h)

Table 1. Breakdown of a test run of the LTA imaging pipeline using the *PROCESS* services and infrastructure

In Table 1, we show a breakdown of the average execution time of each step in the *FACTOR* pipeline. This pipeline is running on a test dataset consisting of twenty sub-bands (out of 144) of both the calibrator and target data, and processes about 450 GB of data. The end-to-end execution time of the pipeline is about 6 days. As the table shows, the processing time is dominated by the direction dependent calibration performed by *FACTOR*, which takes about 89 % of the overall time required.

⁵ https://fits.gsfc.nasa.gov/fits_standard.html

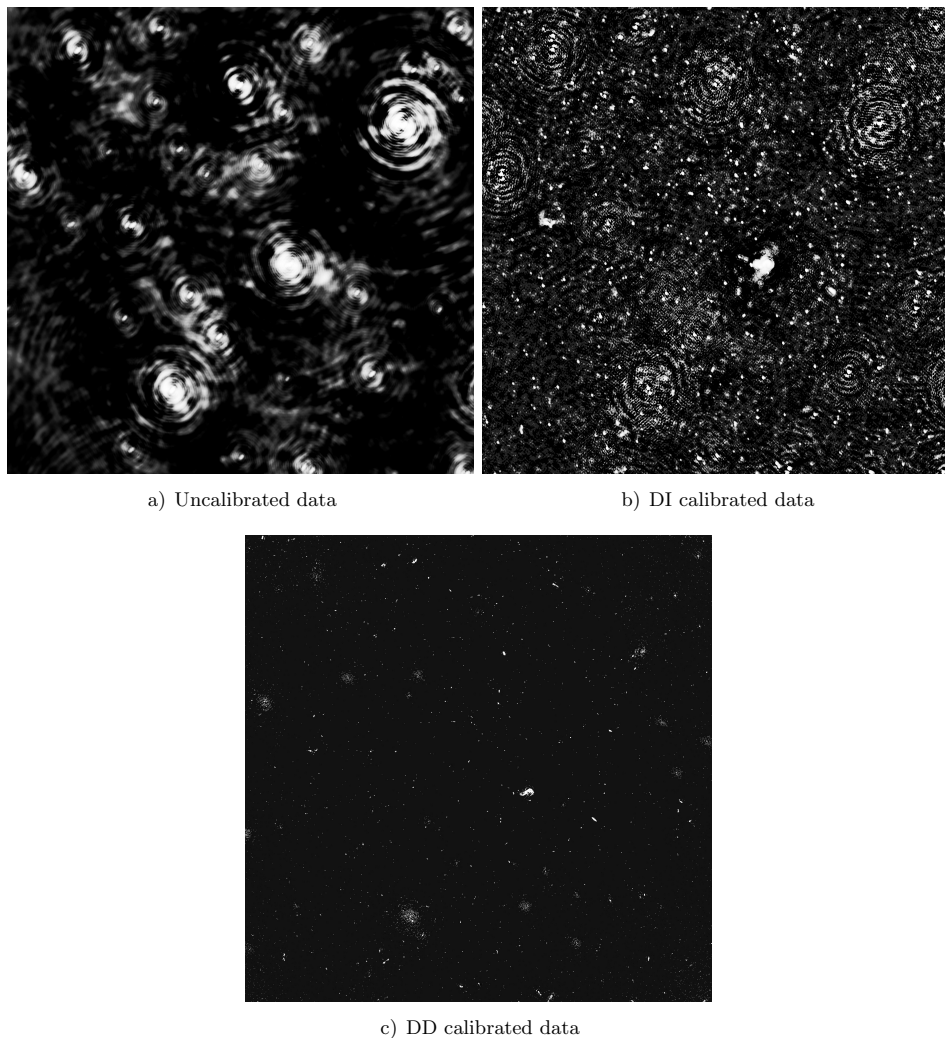


Figure 9. Results of imaging uncalibrated, DI calibrated and DD calibrated data

6 CONCLUSION AND FUTURE WORK

In this paper, we discussed our efforts in “unlocking” the LOFAR LTA using the software ecosystem developed in the PROCESS project. We described the motivation for our use case and analysed its requirements. We succinctly described the science case behind it and briefly presented the PROCESS project services and tools. Then we showed that the solution for our use case can be straightforwardly implemented using the PROCESS services and tools. Finally, we showed an example

of the sky maps generated using that solution and the time needed to reach those results.

One of the nonfunctional requirements identified for our use case is horizontal scalability that allows the processing of several observations in parallel, and potentially on different compute sites. This feature would be very useful for the Surveys KSP for instance, as they typically require a large amount of processing. Although, theoretically, IEE can submit to several computing sites concurrently, in practice, it can currently only submit to its local computing site, Prometheus. As our future work, we hope to extend this to multiple sites.

In addition, all processing is currently performed on a single node. Parallelism is limited to the number of cores available within this node. It would be interesting to revisit these processing steps and reimplement them using more scalable approaches. As the Direction Dependent calibration step is the most compute-intensive, it would be beneficial to add alternative approaches to *FACTOR* such as the DDF pipeline [4] or SAGECal [24], which may support better parallelisation schemes.

Acknowledgements

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.

REFERENCES

- [1] AA-ALERT: Access and Acceleration of the Apertif Legacy Exploration of the Radio Transient Sky. <https://www.esciencecenter.nl/project/aa-alert>, accessed: 2020-09-17.
- [2] Apache Airflow. <https://airflow.apache.org>, accessed: 2020-09-09.
- [3] Cloudify: Multi-Cloud Orchestration. <https://cloudify.co/>, accessed: 2020-09-10.
- [4] Ddf-Pipeline (GitHub). <https://github.com/mhardcastle/ddf-pipeline>, accessed: 2020-09-09.
- [5] The Default Pre-Processing Pipeline (DPPP). <https://support.astron.nl/LOFARImagingCookbook/dppp.html>, accessed: 2020-09-09.
- [6] The European Open Science Cloud for Research Pilot Project. <https://eoscpilot.eu>, accessed: 2020-09-09.
- [7] Factor: Facet Calibration for LOFAR. <https://www.astron.nl/citt/facet-doc/index.html>, accessed: 2020-09-09.
- [8] FRB Catalogue. <http://www.frbcat.org>, accessed: 2020-09-17.
- [9] Generic Pipeline. <https://www.astron.nl/citt/genericpipeline/>, accessed: 2020-09-09.

- [10] The Interactive Execution Environment (IEE). <https://gitlab.com/cyfronet/iee>, accessed: 2020-09-10.
- [11] JSON Schema. <https://json-schema.org/>, accessed: 2020-09-09.
- [12] The LOBCDER Data Services. <https://github.com/micro-infrastructure/mini-lobcdcr>, accessed: 2020-09-10.
- [13] LOFAR Key Science Projects. <http://www.lofar.org/astronomy/key-science/lofar-key-science-projects.html>, accessed: 2020-09-09.
- [14] LOFAR Long Term Archive Pipeline Orchestrate Web Application. <https://github.com/EOSC-LOFAR/ltacat>, accessed: 2020-09-09.
- [15] LOFAR LTA Pipeline Orchestrate Web Application. https://github.com/process-project/ltacat_UC2, accessed: 2020-09-09.
- [16] LOFAR Wiki: Standard Imaging Pipeline. https://www.astron.nl/lofarwiki/doku.php?id=public:user_software:documentation:standard_imaging_pipeline, accessed: 2020-09-09.
- [17] LTA Processing Pipeline Template and Guide. https://github.com/process-project/UC2_pipeline, accessed: 2020-09-09.
- [18] Netherlands Institute for Radio Astronomy. <https://www.astron.nl>, accessed: 2020-09-09.
- [19] Picas Overview: http://doc.grid.surfsara.nl/en/latest/Pages/Practices/picas/picas_overview.html, accessed: 2020-09-09.
- [20] Prefactor: Preprocessing for Facet Calibration for LOFAR. <https://www.astron.nl/citt/prefactor/>, accessed: 2020-09-09.
- [21] PROCESS: Providing Computing Solutions for Exascale Challenges. <https://www.process-project.eu>, accessed: 2020-09-09.
- [22] PROCESS Use Case Descriptions. <https://www.process-project.eu/use-cases>, accessed: 2020-09-09.
- [23] Rimrock – Robust Remote Process Controller. <http://dice.cyfronet.pl/products/rimrock>, accessed: 2020-09-10.
- [24] SAGECal (GitHub). <https://github.com/nlesc-dirac/sagecal>, accessed: 2020-09-09.
- [25] AMSTUTZ, P.—CRUSOE, M. R.—TIJANIĆ, N.—CHAPMAN, B.—CHILTON, J.—HEUER, M.—KARTASHOV, A.—LEEHR, D.—MÉNAGER, H.—NEDELJKOVICH, M.—SCALES, M.—SOILAND-REYES, S.—STOJANOVIC, L.: Common Workflow Language, v1.0. Figshare, 2016, doi: 10.6084/m9.figshare.3115156.v2.
- [26] BOBÁK, M.—HLUCHY, L.—BELLOUM, A. S. Z.—CUSHING, R.—MEIZNER, J.—NOWAKOWSKI, P.—TRAN, V.—HABALA, O.—MAASSEN, J.—SOMOSKÖI, B.—GRAZIANI, M.—HEIKKURINEN, M.—HÖB, M.—SCHMIDT, J.: Reference Exascale Architecture. 2019 15th International Conference on eScience (eScience), San Diego, CA, USA, 2019, pp. 479–487, doi: 10.1109/eScience.2019.00063.
- [27] BUBAK, M.—MEIZNER, J.—NOWAKOWSKI, P.—BOBÁK, M.—HABALA, O.—HLUCHÝ, L.—TRAN, V.—BELLOUM, A. S. Z.—CUSHING, R.—HÖB, M.—KRANZLMÜLLER, D.—SCHMIDT, J.: A Hybrid HPC and Cloud Platform for Multi-disciplinary Scientific Application. 2020 Super Computing Frontiers Europe, Virtual Global Conference, March 2020.

- [28] COHEN, A. S.—LANE, W. M.—COTTON, W. D.—KASSIM, N. E.—LAZIO, T. J. W.—PERLEY, R. A.—CONDON, J. J.—ERICKSON, W. C.: The VLA Low-Frequency Sky Survey. *The Astronomical Journal*, Vol. 134, 2007, No. 3, pp. 1245–1262, doi: 10.1086/520719.
- [29] CORNWELL, T. J.—GOLAP, K.—BHATNAGAR, S.: W Projection: A New Algorithm for Wide Field Imaging with Radio Synthesis Arrays. In: Shopbell, P., Britton, M., Ebert, R. (Eds.): *Astronomical Data Analysis Software and Systems XIV*. Astronomical Society of the Pacific, San Francisco, ASP Conference Series, Vol. 347, 2005, pp. 86–90.
- [30] DE BRUYN, A. G.: The Westerbork Northern Sky Survey. In: Ekers, R., Fanti, C., Padrielli, L. (Eds.): *Extragalactic Radio Sources*. Springer, Dordrecht, International Astronomical Union, Vol. 175, 1996, pp. 495–498, doi: 10.1007/978-94-009-0295-4_180.
- [31] INTEMA, H. T.—JAGANNATHAN, P.—MOOLEY, K. P.—FRAIL, D. A.: The GMRT 150 MHz All-Sky Radio Survey – First Alternative Data Release TGSS ADR1. *Astronomy and Astrophysics*, Vol. 598, 2017, Art. No. A78, 28 pp., doi: 10.1051/0004-6361/201628536.
- [32] KURTZER, G. M.—SOCHAT, V.—BAUER, M. W.: Singularity: Scientific Containers for Mobility of Compute. *PLoS ONE*, Vol. 12, 2017, No. 5, Art. No. e0177459, 20 pp., doi: 10.1371/journal.pone.0177459.
- [33] MECHEV, A. P.—OONK, J. B. R.—DANEZI, A.—SHIMWELL, T. W.—SCHRIJVERS, C.—INTEMA, H. T.—PLAAT, A.—RÖTTGERING, H. J. A.: An Automated Scalable Framework for Distributing Radio Astronomy Processing Across Clusters and Clouds. *International Symposium on Grids and Clouds 2017 (ISGC 2017)*, Academia Sinica, Taipei, Taiwan, 2017, Art. No. 002. <https://pos.sissa.it/293/002/pdf>.
- [34] MECHEV, A. P.—OONK, J. B. R.—SHIMWELL, T.—PLAAT, A.—INTEMA, H. T.—RÖTTGERING, H. J. A.: Fast and Reproducible LOFAR Workflows with AGLOW. *2018 IEEE 14th International Conference on e-Science (eScience)*, Amsterdam, Netherlands, 2018, Vol. 1, pp. 136–144, doi: 10.1109/eScience.2018.00029.
- [35] OONK, J.—MECHEV, A.—DANEZI, A.—SCHRIJVERS, C.—SHIMWELL, T.: Radio Astronomy on a Distributed Shared Computing Platform: The LOFAR Case. 2017.
- [36] RENTING, G. A.—HOLTIES, H. A.: LOFAR Long Term Archive. In: Evans, I. N., Accomazzi, A., Mink, D. J., Rots, A. H. (Eds.): *Astronomical Data Analysis Software and Systems XX*. Astronomical Society of the Pacific, San Francisco, ASP Conference Series, Vol. 442, 2011, pp. 49–52.
- [37] SABATER, J.—SÁNCHEZ-EXPÓSITO, S.—BEST, P.—GARRIDO, J.—VERDES-MONTENEGRO, L.—LEZZI, D.: Calibration of LOFAR Data on the Cloud. *Astronomy and Computing*, Vol. 19, 2017, pp. 75–89, doi: 10.1016/j.ascom.2017.04.001.
- [38] SHIMWELL, T. W.—TASSE, C.—HARDCASTLE, M. J.—MECHEV, A. P.—WILLIAMS, W. L.—BEST, P. N.—RÖTTGERING, H. J. A.—CALLINGHAM, J. R.—DIJKEMA, T. J.—DE GASPERIN, F. et al.: The LOFAR Two-Metre Sky Survey. II. First Data Release. *Astronomy and Astrophysics*, Vol. 622, 2019, Art. No. A1, 21 pp., doi: 10.1051/0004-6361/201833559.

- [39] TASSE, C.—VAN DER TOL, S.—VAN ZWIETEN, J.—VAN DIEPEN, G.—BHATNAGAR, S.: Applying Full Polarization A-Projection to Very Wide Field of View Instruments: An Imager for LOFAR. *Astronomy and Astrophysics*, Vol. 553, 2013, Art. No. A105, 13 pp., doi: 10.1051/0004-6361/201220882.
- [40] VAN HAARLEM, M. P.—WISE, M. W.—GUNST, A. W.—HEALD, G.—MCKEAN, J. P.—HESSELS, J. W. T.—DE BRUYN, A. G.—NIJBOER, R.—SWINBANK, J.—FALLOWS, R. et al.: LOFAR: The LOw-Frequency ARray. *Astronomy and Astrophysics*, Vol. 556, 2013, Art. No. A2, 53 pp., doi: 10.1051/0004-6361/201220873.
- [41] VAN WEEREN, R. J.—WILLIAMS, W. L.—HARDCASTLE, M. J.—SHIMWELL, T. W.—RAFFERTY, D. A.—SABATER, J.—HEALD, G.—SRIDHAR, S. S.—DIJKEMA, T. J.—BRUNETTI, G. et al.: LOFAR Facet Calibration. *The Astrophysical Journal Supplement Series*, Vol. 223, 2016, No. 1, Art. No. 2, 16 pp., doi: 10.3847/0067-0049/223/1/2.



Souley MADOUGOU is 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.



Hanno SPREEUW is an eScience Research Engineer at the Netherlands eScience Center since February 2015. His Ph.D. research paved the way for the detection of transient radio sources with LOFAR. During his subsequent postdoc position at the Netherlands Cancer Institute he accelerated CPU code for 3D dose reconstruction from radiotherapy treatments in real time. At the Netherlands eScience Center, his projects mostly involve astronomy or physics with a focus on accelerated computing using GPUs.



Jason MAASSEN is Technology Lead at the Netherlands eScience Center. He is involved in many of the projects at the center that 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.

TOWARDS EXASCALE COMPUTING ARCHITECTURE AND ITS PROTOTYPE: SERVICES AND INFRASTRUCTURE

Jan MEIZNER, Piotr NOWAKOWSKI

*ACC Cyfronet, AGH University of Science and Technology
Krakow, Poland*

✉

*Sano Centre for Computational Medicine
Krakow, Poland*

e-mail: {j.meizner, p.nowakowski}@cyfronet.pl

Jan KAPALA, Patryk WOJTOWICZ

*ACC Cyfronet, AGH University of Science and Technology
Krakow, Poland*

e-mail: {j.kapala, p.wojtowicz}@cyfronet.pl

Marian BUBAK

*ACC Cyfronet, AGH University of Science and Technology
Krakow, Poland*

✉

*Sano Centre for Computational Medicine
Krakow, Poland*

✉

*Department of Computer Science, AGH University of Science and Technology
Krakow, Poland*

e-mail: bubak@agh.edu.pl

Viet TRAN, Martin BOBÁK

*Institute of Informatics, Slovak Academy of Sciences
Dúbravská cesta 9, 845 07 Bratislava, Slovakia*

e-mail: {viet.tran, martin.bobak}@savba.sk

Maximilian HÖB

*Munich Network Management Team (MNM-Team)
Ludwig-Maximilians Universität, Munich, Germany
e-mail: hoeb@nm.ifi.lmu.de*

Abstract. This paper presents the design and implementation of a scalable compute platform for processing large data sets in the scope of the EU H2020 project PROCESS. We are presenting requirements of the platform, related works, infrastructure with focus on the compute components and finally results of our work.

Keywords: Exascale computing, large data sets, HPC, cloud computing

1 INTRODUCTION

Despite continuous increases in the computing power of HPC systems, even the largest system on the latest edition of the TOP500 list [20] still provides less than 0.5 exaFLOP (in full precision). At the outset of the PROCESS project this figure was even lower (about 0.1 exaFLOP) [19]. It therefore seemed clear to us that – at least initially – reaching exascale computing capabilities would require a combination of multiple HPC systems. As we can see, this premise holds true to this day, and even when we finally break the exaFLOP barrier, the number of such systems will be highly limited – thus, combining the power of many sites would remain prudent in many cases. It is also important to notice that it is not always possible to quickly move data between computing sites; hence, the ability to bring computations to data remains an important issue.

Running tasks in such combined systems is fraught with multiple challenges. The basic one relates to the heterogeneity of the environment, as each system is operated by a different entity. This heterogeneity may involve access mechanisms (protocols, credential types) as well as software installed on clusters (OS type and version, queuing system). Additionally, each cluster usually runs its workloads in internal private LANs which do not allow inbound connections from the Internet (due to mechanisms such as NAT). This, in turn, places additional restrictions upon the designed infrastructure, such as the lack of direct P2P communication between jobs running on different clusters. One of the main goals of the project was to provide a mechanism which would allow running jobs on multiple sites and move data around freely.

In order to manage such heterogeneous systems, it is also crucial [14] to provide a highly scalable platform, able to process large quantities of data (although issues related to maintenance of such data have been described in a separate paper [23]).

Finally, we need to take into account that some sites in a distributed computational environment may be based on traditional HPC paradigms (such as the Prometheus cluster at Cyfronet, or CoolMUC and SuperMUC-NG at Leibniz-Rechenzentrum) while others may be Cloud-based (in the presented case, this includes the Institute of Informatics of the Slovak Academy of Sciences and other compatible private and community clouds such as those provided in the scope of the European Open Science Cloud; EOSC). This further elevates the level of heterogeneity [8].

2 A SHORT OVERVIEW OF RESEARCH ON EXASCALE SYSTEM

A systematic analysis of needs and profits of exascale computing systems was initiated by the U.S. DOE in a series of workshops on scientific grand challenges in 2007 [22]. Workshops were focused on the grand challenges of specific scientific domains and on the role for scientific computing in addressing those challenges. A summary of discussions during those scientific meeting is presented in [6]. It also gives an initial set of requirements for exascale systems, however, it is mostly concentrated on the computing aspect leaving aside big data aspects. Dongarra et al. in [9] analysed the approaches used to implement peta- and exa-scale computing pointing out that completely uncoordinated development model will not provide the software needed to support the unprecedented parallelism required for peta/exascale computation and presented the idea of the International Exascale Software Project. In this paper, the development of appropriate open-source software tools was clearly indicated as an important factor in quest for high performance and productivity. The focus was mostly on proper usage of new processors architectures of large parallel computers.

The need for considering together exascale computing and big data was presented by Reed and Dongarra in [18]; after overview of main area for exascale computing like biology, particle physics, climate science, cosmology, astrophysics, material science, they have analysed technical challenges in advanced computing including software elaboration, and overview a number of national and international projects. The ideas presented in this paper are reflected in the Big Data and Extreme Scale computing Project [1] and in the ECP – Exascale Computing Project [2]. On the basis of the achievements of these projects, an international group of scientists [5] elaborated a set of recommendations for successful approaches for software ecosystem convergence in big, logically centralized facilities that execute large-scale simulations and models and/or perform large-scale data analytics. In our research, we are going to take them into account having in mind that our solutions should be appropriate for exascale computing and data analytics on distributed systems.

The recently published paper [16] presents results of a study of possible convergence of big data coming from distributed scientific instruments and sensors and high performance computing appropriate for coming era of next-generation data centric computing. The study was performed in the framework of the EU project SAGE. The authors have elaborated an advanced storage system which has been

implemented and installed at the Jülich Supercomputing Center. It is a very interesting and important solution, however, it does not address usage of distributed computing resources.

3 REQUIREMENTS

The ultimate goal of PROCESS is to provide a versatile solution well suited for a wide range of use cases from multiple domains, both scientific and business-oriented. To achieve this goal we had to collect a set of requirements from each use case, and combine them into an integrated set applicable to all of them.

3.1 Hardware Requirements

The hardware resources are basis for any IT system. In this subsection we are presenting the set of hardware-related requirements such as providing sufficient access to: HPC resources, Cloud resources, accelerated computing resources (including the GPGPU), external infrastructure reachable via API, data storage on the order of 1 PB (distributed).

3.2 Software Requirements

As hardware alone is not sufficient to provide the services sought by scientists, our platform also has to support a wide range of software tools. The common tools for machine learning and deep learning are essential for wide range of medical applications, but also others such as business oriented. Python development environment with support for Jupyter notebooks allows simplification of the Use Case codes developments. Apache Spark, Hadoop and HBase frameworks are necessary to process the big data sets. Support for containers such as Docker and Singularity allows packaging of the codes alongside required dependences for streamlined deployment. Secure access to and extraction from external data resources is of course also crucial for any use case. Finally, we need to provide appropriate support for programming languages and tools needed by the provided use cases' codes such as: Java environment, Grid support, Matlab environment, Large-scale modelling, Predictive analytic methods, Probabilistic risk calculation tools.

3.3 Execution Models

We have analyzed all these requirements and in our conclusions got a set of specific Execution Models, namely: Deep Learning, Exascale Data Management, Exascale Data Extraction, Probabilistic Analysis, Calibration and finally the Pre- and Post-Processing.

All those models had to be reflected in the the PROCESS system architecture – both general as well as Compute parts specific.

4 OVERVIEW OF COMPUTING COMPONENTS

While preparing such a complex system we have carefully taken into account the current state of the art in various respects, relevant to the integrated system and its computing components in particular. This analysis is presented in the following subsections.

4.1 Interactive Execution Environment

Several solutions already exist to support interactive execution of large-scale computations on hybrid underlying infrastructures. This subsection provides a basic description of the available mechanisms and tools which support creation and sharing of executable documents for data analysis. In this section we also provide a brief introduction to scripting notebooks, in particular their integration with HPC infrastructures in order to support building extreme large computing services as well as their extensions mechanisms needed to add support specific to the PROCESS project. We also present the results of comparison of their functionality. This section partially extends the work submitted to KUKDM 2018 conference [13].

During our studies we have analysed multiple notebook-based solutions, a summary of which is presented in Table 1.

Name	Large Data Sets	Infrastructure
R Notebook	Using custom libraries (e.g. for Apache SPARK)	Using custom libraries (e.g. communicating with HPC queuing systems)
DataBricks	The whole platform is based on Apache SPARK	Available only on AWS or Azure
Beaker	Using additional custom libraries	No specific support for HPC; docker version available
Jupyter	Using additional custom libraries	No mature solution for HPC; docker version available
Cloud Datalab	Support for Google data services (e.g. BigQuery, Cloud Machine Learning Engine, etc.)	Only GCP
Zeppelin	Native support for Apache Spark	Possible to run on HPC using connection to YARN cluster

Table 1. Interactive execution environment comparison

Although there exist many interactive execution environments that could be considered for extension to match PROCESS requirements, many also have important drawbacks. DataBricks and Cloud Datalab require to be run on specific cloud resources. Zeppelin and DataBricks are based on the Apache SPARK solution which potentially limits their usage to that platform. RNotebooks seems to be promising,

however some important features are only available with the commercial version of Rstudio. BeakerX (the successor to Beaker) and Cloud Data are actually based on the Jupyter solution, which appears to be the most popular base for building such environments.

The goal of the Interactive Execution Environment is to bridge the gap between users of computational services (who are not expected to be familiar with the complexity of developing and executing extreme large scale computational tasks with the use of modern HPC infrastructures) and the underlying hardware resources. Accordingly, the IEE is envisioned as an interface layer where applications can be accessed and their results browsed in a coherent manner by domain scientists taking part in the PROCESS project and beyond.

The following properties are regarded as particularly desirable:

- A means of implementing the “focus on services and forget about infrastructures” concept;
- Providing two ways of accessing the underlying computational resources: through a user-friendly GUI and programmatically, via a dedicated RESTful API;
- Embeddability in an external environment (such as Jupyter) via API integration;
- Interfacing computational clouds and traditional HPC (batch job submission) and public cloud access libraries, as appropriate.

The features which need to be provided by the environment are as follows:

- Deployment of computational tasks on the available resources,
- Infrastructure monitoring services,
- User-friendly access to PROCESS datasets,
- Security management (users, groups, roles),
- Administrative services (billing and logging),
- Integration with external tools via standardized APIs.

Following discussions with use case developers and the project’s architecture team, the following tools, described further in this paper, have been identified as usable in the context of the PROCESS project – in addition to the previously discussed notebook solutions, which can function as an embedded feature in a comprehensive GUI.

4.2 EurValve Model Execution Environment

The EurValve [7] Model Execution Environment (shown in Figure 1) is an execution environment for data processing pipelines. Originally conceived in the context of the EurValve project, the goal was to develop a decision support system for procedures related to heart valve abnormalities, enabling clinicians to decide upon the optimal

course of action for any specific patient (i.e. choose between medication/surgery and advise on the possible strategies and outcomes of the latter). While the aforementioned DSS does not, by itself, involve large-scale processing, it is based on a knowledge base whose production is one of the principal goals of EurValve. Assembling this knowledge base calls for processing a vast array of patient data (for both prospective and retrospective patients) through the use of dedicated pipelines, consisting of multiple services and making use of various types of supercomputing infrastructures (both traditional batch HPC and cloud models).

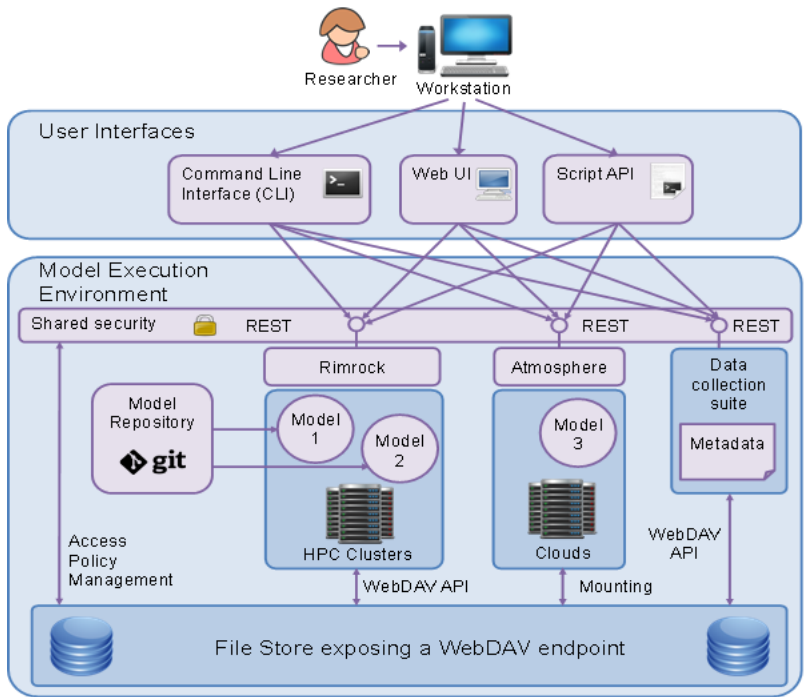


Figure 1. EurValve Model Execution Environment (MEE) architecture

4.3 Rimrock Execution Environment

Rimrock stands for Robust Remote Process Controller Controller [4] shown in Figure 2 is a service that simplifies interaction with remote HPC servers. It can execute applications in batch mode or start an interactive application, where output can be fetched online and new input sent using a simple REST interface. What is more, by using a dedicated REST interface users are able to submit new jobs to the infrastructure. This solution would support efficient creation and sharing of executable documents for analysis of heterogeneous research datasets. RIMROCK is

currently actively used in production in the Polish nationwide HPC infrastructure called PLGrid [17].

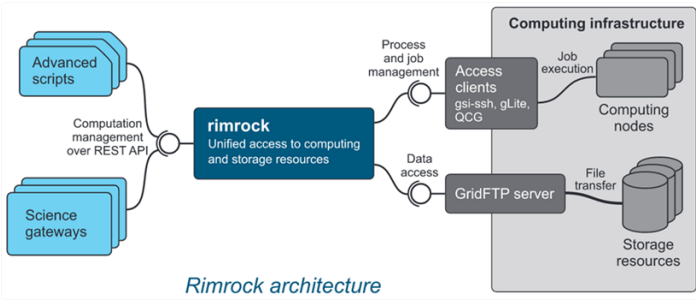


Figure 2. Rimrock architecture

4.4 Atmosphere Cloud Platform

Atmosphere [3], shown in Figure 3, is a hybrid cloud environment facilitating development and sharing of computational services wrapped as cloud virtual machines, with access to external data sources. Atmosphere supports the entire cloud service development lifecycle and provides a set of pluggable interfaces which can be included in portals and web applications. It is compatible with a wide range cloud middleware packages from open-source and commercial vendors, and provides interfaces to both public and private cloud resources in the form of a technology-agnostic UI and RESTful APIs.

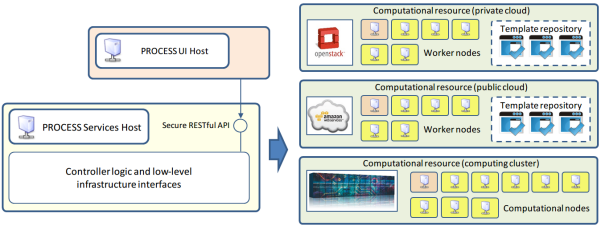


Figure 3. The architecture of atmosphere

4.5 Benchmarking and Monitoring Services

We have analyzed multiple monitoring solutions including Nagios, Zabbix, Icinga, Munin, Cacti, Ganglia, Collectd, Elastic Stack, Grafana, NFDUMP with NfSen and ntopng.

While detailed analysis is out of scope of this paper, we initially identified three candidates for this task: Zabbix, Nagios and Icinga, as they are all mature solutions and possess the required properties. Other available solutions, while usable, would only address a subset of the presented requirements; consequently, their usage would require integration of multiple distinct components, with its attendant impact on the stability of the integrated solution deployed in PROCESS.

Upon further analysis we finally chose Zabbix as the best solution (shown in Figure 4). Icinga 2 (current version of Icinga) was rejected due to not being as mature and well-established as either Zabbix or Nagios. As for Nagios Core, it offers great alerting capabilities, however it does not provide the same level of support for online resource monitoring, and its configuration is based on text files, requiring more complex integration. In contrast, Zabbix’s configuration is based on an RDBMS (like MariaDB/MySQL) and could be modified via an API. With over 19 years of history and constant development/testing, as well as a wide range of users including large corporations representing a wide spectrum of domains such as banking/finance, health, IT and telecommunication, Zabbix is regarded as a mature solution.

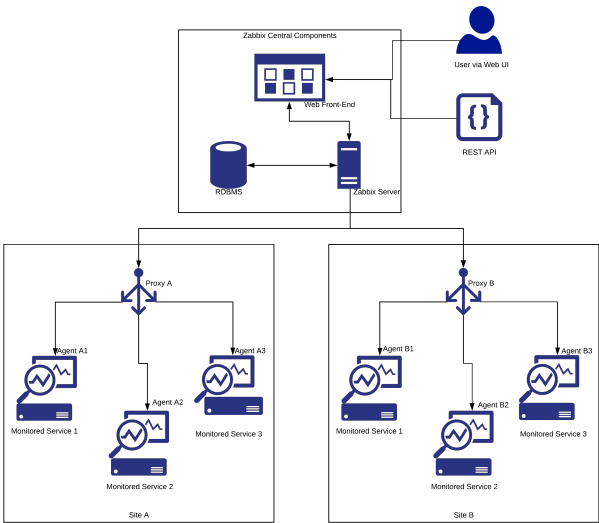


Figure 4. Monitoring architecture

5 CONCEPT OF COMPUTING ARCHITECTURE

The overall architecture of the PROCESS Platform [15] is presented in a separate paper [24]. In this section we focus solely on the computing components and glue code connecting them to one another as well as to external services.

While this aspect represents only part of the bigger platform, due to the inherent complexity of an exascale-capable system we had to design a solution composed of multiple layers, namely:

- API for third-party services (like external Portals),
- Web UI for domain scientists (to configure pipelines),
- Connectors for HPC and Cloud backend APIs,
- The aforementioned backend APIs themselves (Rimrock [4] and Cloudify [10]),
- Containers suitable for specific HPC/Cloud resources [12],
- Underlying HPC and Cloud e-Infrastructures.

6 BUILDING THE COMPUTING PLATFORM

6.1 IEE

The Interactive Execution Environment (IEE) is a platform which enables execution of HPC applications – including the PROCESS pilot use cases – in a heterogeneous infrastructure which comprises multiple computing sites based on various computing paradigms (such as “classic” batch-oriented HPC, interactive cloud computing and more). The basic architecture of the environment is schematically depicted in Figure 5. The IEE user interface serves as the entry point for the platform. Computations which rely on batch access to HPC sites are processed by a dedicated service called Rimrock, which can delegate operations to the underlying resources while exposing a RESTful API for IEE (and other tools) to use. In addition, IEE features integration with the Cloudify cloud platform, which enables scheduling cloud VMs from it.

In the context of validating the PROCESS architecture (and that of IEE in particular), we decided to focus on the full pipeline for the use case which involves processing the LOFAR dataset. This enables the means to select the HPC Site and then facilitate the process of staging in data from the LOFAR Long Term Archives and moving it to the relevant site, running relevant computation using site-specific settings (Queuing and Container systems) and finally staging out the results.

For this case we decided to utilize two separate HPC sites: the Prometheus supercomputing cluster at ACC Cyfronet AGH in Kraków (same as for the first prototype), as well as the SuperMUC-NG Cluster at LRZ in Garching bei München. In addition to those HPC sites, to showcase the full capabilities of the platform, we also included a demo of an additional application from another use case (Ancillary pricing for airline revenue management) deployed on the OpenStack cloud at UISAV in Bratislava via the Cloudify component.

Use case applications are organized as projects composed of multiple steps, each of which involves either processing (computations) or is related to data transfer (in particular, staging in the relevant data using the PROCESS data storage infrastructure, or accessing results of computational tasks for visualization and download).

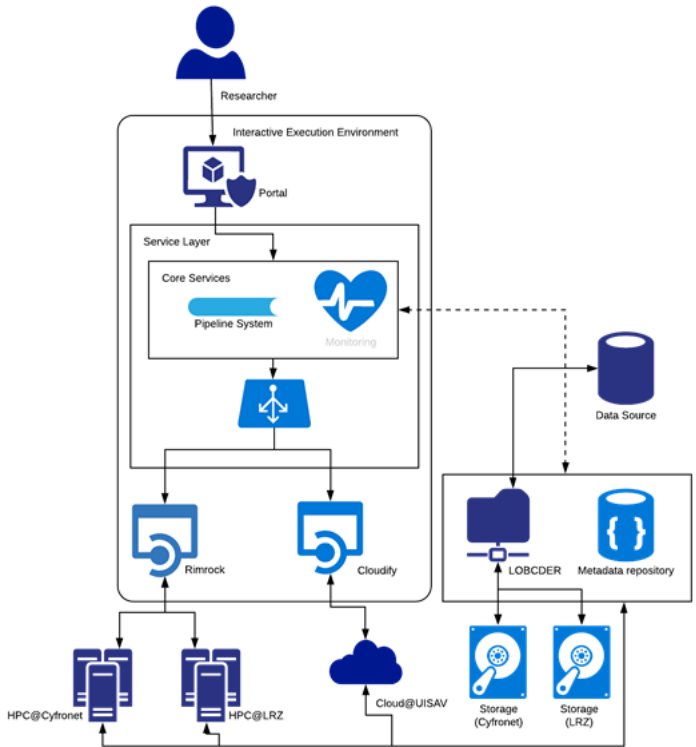


Figure 5. Schematic depiction of the IEE architecture, including its integration with HPC and data storage sites

6.2 Rimrock

The Rimrock component enables running batch scripts on an HPC infrastructure via a convenient REST API. In the scope of the PROCESS project we use it to spawn relevant use case computational components in the form of containers. The container technology is proper for each site, so in the case of Prometheus it is Singularity and in the case of SuperMUC-NG it is Charlie Cloud [21].

A schematic overview of the Rimrock architecture is presented in Figure 7. The service is invoked from the IEE component described in Section 6.1.

6.3 Cloudify Orchestration Service

Service orchestration is often understood as the process of automated configuration, deployment and other management activities of services and applications in the cloud. It can automate execution of different service workflows, including deploy-

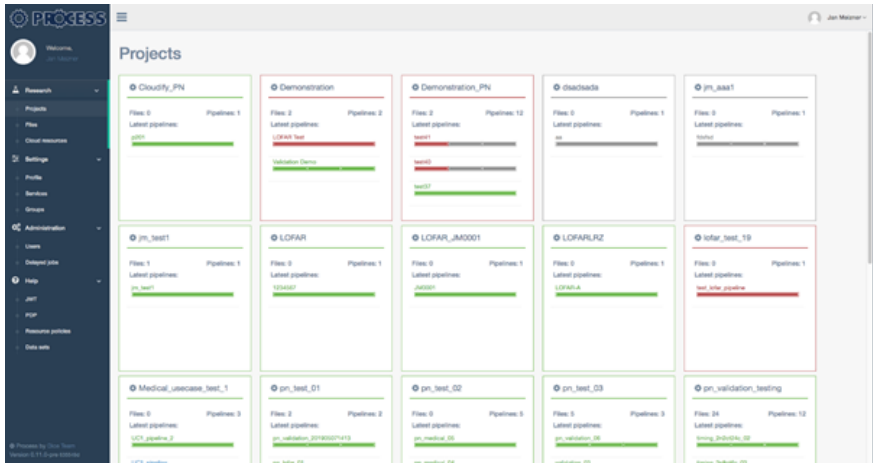


Figure 6. IEE user workbench

ment, initialization, start/stop, scaling, healing of services based on standardized descriptions of composed services, relations between components and their requirements. In the PROCESS project, we use the OASIS TOSCA standard for service description and Cloudify for orchestration.

Cloudify (<https://docs.cloudify.co/4.5.0/about/>) is an open source cloud orchestration platform, designed to automate the deployment, configuration and remediation of application and network services across hybrid cloud and stack environments. It uses OASIS TOSCA templates written in YAML (called blueprints in Cloudify) for defining applications, services and dependences among them. These blueprint files describe also the execution plans for the lifecycle of the application for installing, starting, terminating, orchestrating and monitoring the application stack. Cloudify uses the blueprint as input that describes the deployment plan and is responsible for executing it on the cloud environment. Figure 8 shows the architecture of the Cloudify orchestration service.

Cloudify has its own console, see Figure 9, but for integrating with other services, it is more comfortable using its REST API. Cloudify has completed REST API for all operations related to service orchestration. This REST API can be divided into several sections, the most important ones are the following:

Blueprint: management of TOSCA templates, e.g. upload, download, list, delete.

Deployment: deployment of services and management of already deployed services.

Execution: executing workflows defined in TOSCA templates on concrete deployment, e.g. install, restart, uninstall.

The details of REST API is described at <https://docs.cloudify.co/4.5.0/developer/apis/rest-service/>.

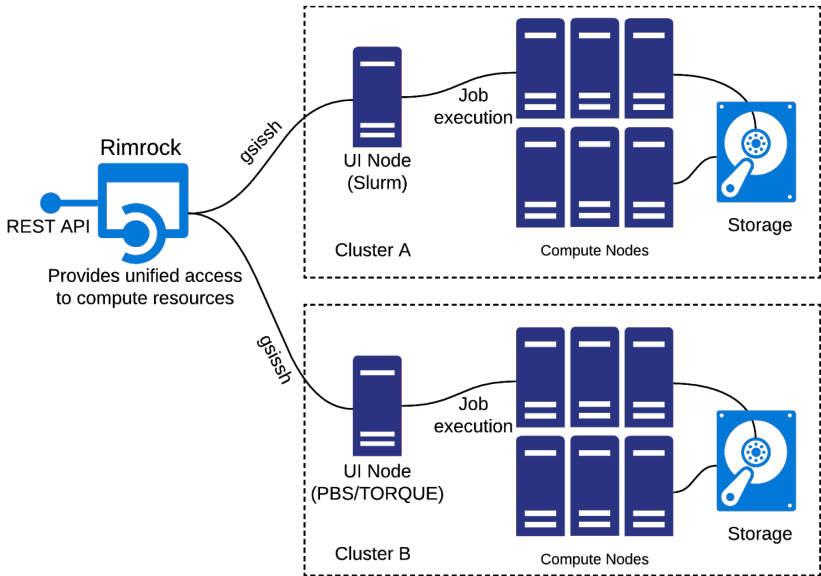


Figure 7. Schematic depiction of Rimrock, including Compute and Storage resources

The API is used by command-line clients (CLI) or scientific gateways (GUI) for deployment and management of services. The services need to be described in TOSCA templates (blueprints) and uploaded to the Cloudify server before deployment using blueprint API. After that, the users can deploy/undeploy instances of services described in the blueprints via deployment API or execute a specific workflow, e.g. restart the service running in the cloud.

Currently Cloudify is integrated with the data micro-infrastructure and IEE. A blueprint has been created for dynamic deployment of new nodes in the cloud and adding them to the Kubernetes cluster for the data micro-infrastructure. The use case of Ancillary pricing for airline revenue management runs in the Cloud using the Cloudify orchestration service. By communicating with the Cloudify API, IEE can manage the execution of the use case in the Cloud.

7 PLATFORM USAGE BY THE USE CASES

In this section we present the ways in which the PROCESS computing platform has been integrated with the use cases considered in the project.

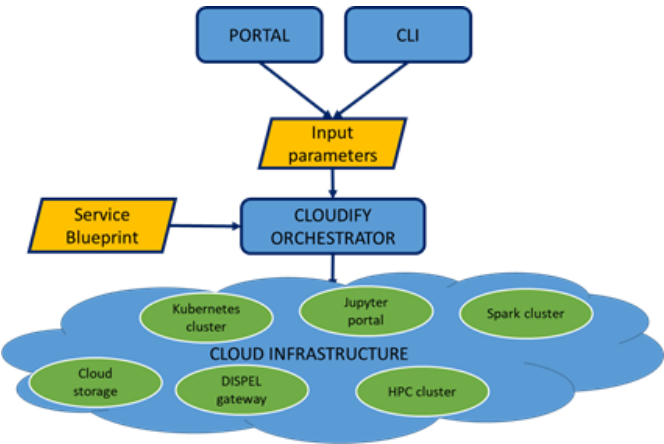


Figure 8. Architecture of Cloudify orchestration service

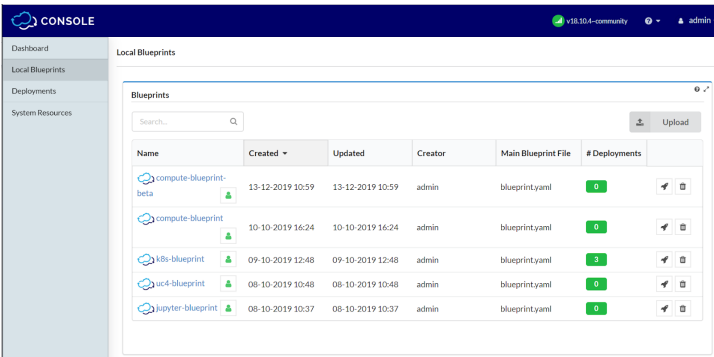


Figure 9. Cloudify console

7.1 Content-Based Search and Classification

The goal of this use case is to improve performance of AI-based medical image analysis using GPU-accelerated distributed computing backed by HPC resources. As the original code was prepared as a Docker container, part of our work was to port it to the Singularity format which is designed for multi-tenant environments.

The basic workflow used for this use case is presented in Figure 10.

7.2 Square Kilometre Array (SKA)

The SKA use case goal is to prepare the computational platform and domain codes for extreme challenges of the SKA radiotelescope when it is available, by using existing datasets procured using the LOFAR telescope.

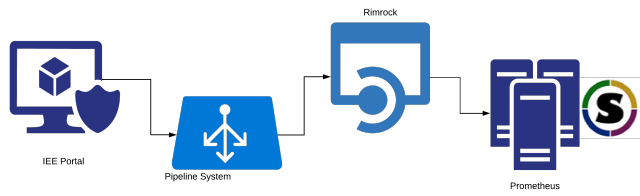


Figure 10. Content-based search and classification use case

Given that LOFAR produces large volumes of data even in its present state, and that this data is stored in multiple locations, this use case obviously presents a challenge for the data transfer subsystem, but at the same time sufficient computational resources must be provided to quickly process incoming data, as well as enable multi-site execution to bring computation nearer to the data.

Additionally, as the process of querying for the right data is complex and requires broad domain knowledge, we have decided to provide scientists who possess such knowledge with a familiar environment. To this end we have undertaken an effort to integrate the existing LOFAR Portal with IEE using a specialized REST API, as shown in Figure 11.

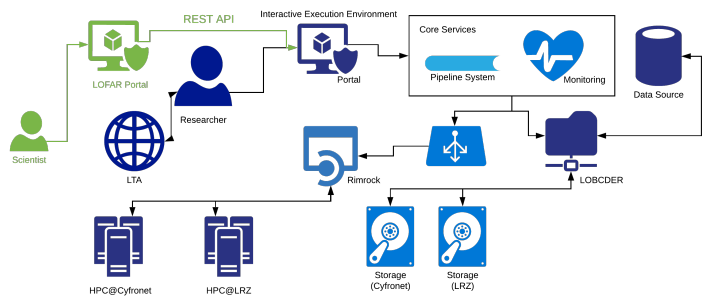


Figure 11. Square Kilometre Array (SKA) use case

7.3 Ancillary Pricing for Airline Revenue Management

This use case differs a bit from the others as it is a business-oriented one. This imposes additional constraints on the compute platform such as the requirement to build a platform that could be easily reproduced in a commercial environment.

To this end we have decided to provide a platform based on cloud resources. The service was deployed using the cloud infrastructure in Slovakia, however such

an environment can be adapted to work with other cloud providers to fulfill the said requirements. The infrastructure utilizes the Cloudify component, as shown in Figure 12.

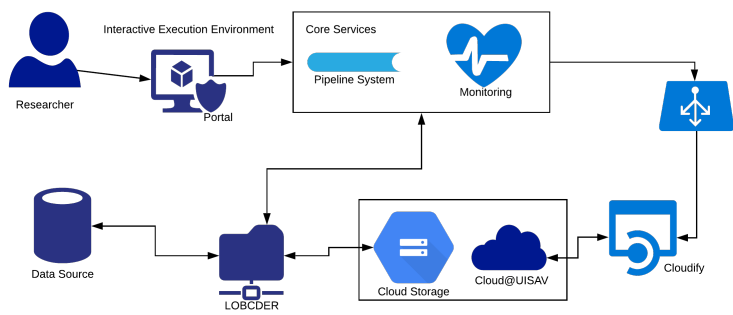


Figure 12. Ancillary pricing for airline revenue management use case

7.4 Agro-Copernicus

Finally, Agro-Copernicus is an example of a use case that features the use of a proprietary component called PROMET, which, due to licensing restrictions, cannot be directly accessed in a fashion similar to other use case codes. The only access mechanism is available via the provided REST API used to control computation, where the code itself is treated as a black box. This solution is shown in Figure 13.

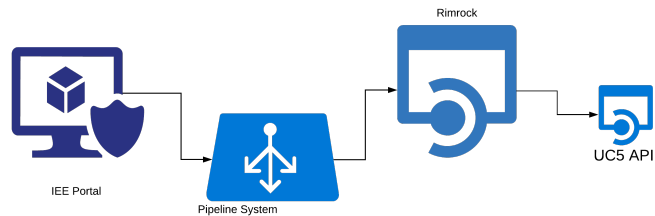


Figure 13. Agro-Copernicus use case

The mechanism can be reused for a wide range of software delivered in the Software as a Service model, as long as the proper API is available.

8 CONCLUSIONS AND FUTURE WORK

In this paper we have presented the work performed in the scope of the PROCESS Project from the beginning until close to its conclusion. This includes state of the art analysis, requirement gathering, platform design and implementation, and, finally, integration with use case codes.

The platform has already been validated using four different use cases, but it is also capable of providing support for other applications, whether based on traditional HPC resources, computing clouds or external services exposing proper APIs.

By the end of the project we aim to provide a unified and straightforward mechanism enabling deployment of all platform components to a containerized environment. In the future we will also seek to further extend the range of supporting cases, as well as make the platform ready for even more powerful upcoming infrastructures.

Within the project we will also continue to follow the containerization approach for scalable HPC applications and will integrate the EASEY framework described in [11], which enables also non-computing experts to easily deploy their applications inside a Charliecloud container through the PROCESS ecosystem.

Acknowledgements

This work is supported by the “PROviding Computing solutions for ExaScale ChallengeS” (PROCESS) project that received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 777533. This research was supported in part by PL-Grid Infrastructure. This work is supported 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] Big Data and Extreme Scale Computing Project. Accessed Oct 26, 2020, available at: <https://www.exascale.org/bdec/>.
- [2] ECP – Exascale Computing Project. Accessed Oct 26, 2020, available at: <https://www.exascaleproject.org/reports/>.
- [3] ACK Cyfronet AGH, DICE Team: Atmosphere. 2020, accessed Sep 14, 2020, available at: <http://dice.cyfronet.pl/products/atmosphere>.
- [4] ACK Cyfronet AGH, DICE Team: Rimrock. 2020, accessed Sep 14, 2020, available at: <https://submit.plgrid.pl>.
- [5] ASCH, M.—MOORE, T.—BADIA, R.M.—BECK, M.—BECKMAN, P.H.—BIDOT, T.—BODIN, F.—CAPPELLO, F.—CHOUDHARY, A.N.—DE SUPINSKI, B.R. et al.: Big Data and Extreme-Scale Computing: Pathways to Convergence – Toward a Shaping Strategy for a Future Software and Data Ecosystem for

- Scientific Inquiry. International Journal of High Performance Computing Applications, Vol. 32, 2018, No. 4, pp. 435–479, doi: 10.1177/1094342018778123.
- [6] 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.
 - [7] BUBAK, M.—BARTYŃSKI, T.—GUBALA, T.—HAREŻŁAK, D.—KASZTELNIK, M.—MALAWSKI, M.—MEIZNER, J.—NOWAKOWSKI, P.: EurValve Model Execution Environment in Operation. In: Turała, M., Bartyński, T., Wiatr, K. (Eds.): CGW Workshop '17, Proceedings. Academic Computer Centre CYFRONET AGH, 2017, pp. 65–66.
 - [8] BUBAK, M.—MEIZNER, J.—NOWAKOWSKI, P.—BOBÁK, M.—HABALA, O.—HLUCHÝ, L.—TRAN, V.—BELLOUM, A. S. Z.—CUSHING, R.—HÖB, M.—KRANZLMÜLLER, D.—SCHMIDT, J.: A Hybrid HPC and Cloud Platform for Multidisciplinary Scientific Application. 2020, <https://www.process-project.eu/wp-content/uploads/2020/03/SCFE2020-PROCESS-23-03-2020.pdf>.
 - [9] DONGARRA, J.—BECKMAN, P.—MOORE, T.—AERTS, P.—ALOISIO, G.—ANDRE, J.-C.—BARKAI, D.—BERTHOU, J.-Y.—BOKU, T.—BRAUNSCHWEIG, B. et al.: The International Exascale Software Project Roadmap. International Journal of High Performance Computing Applications, Vol. 25, 2011, No. 1, pp. 3–60, doi: 10.1177/1094342010391989.
 - [10] GigaSpaces Technologies, Inc.: Cloudify. 2020, accessed Oct 7, 2020, available at: <https://cloudify.co/>.
 - [11] HÖB, M.—KRANZLMÜLLER, D.: Enabling EASEY Deployment of Containerized Applications for Future HPC Systems. In: Krzhizhanovskaya, V. et al. (Eds.): Computational Science – ICCS 2020. Springer, Cham, Lecture Notes in Computer Science, Vol. 12137, 2020, pp. 206–219, doi: 10.1007/978-3-030-50371-0.15.
 - [12] MEIZNER, J.—BUBAK, M.—KAPALA, J.—NOWAKOWSKI, P.—WÓJTOWICZ, P.: Use of the HPC Containers in the Way Towards Exascale. In: Wiatr, K., Bubak, M., Turała, M. (Eds.): CGW Workshop '18. Academic Computer Centre CYFRONET AGH, 2018, pp. 21–22.
 - [13] RYCERZ, K.—NOWAKOWSKI, P.—MEIZNER, J.—WILK, B.—BUJAS, J.—JARMOCIK, L.—KROK, M.—KURC, P.—LEWICKI, S.—MAJCHER, M.—OCIEPKA, P.—PETKA, L.—PODSIADŁO, K.—SKALSKI, P.—ZAGRAJCZUK, W.—ZYGMUNT, M.—BUBAK, M.: A Survey of Interactive Execution Environments for Extreme Large-Scale Computations. KDM '18, Zakopane, Poland, 2018.
 - [14] 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.
 - [15] BOBÁK, M.—HLUCHÝ, L.—BELLOUM, A. S. Z.—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.
- [16] NARASIMHAMURTHY, S.—DANILOV, N.—WU, S.—UMANESAN, G.—MARKIDIS, S.—RIVAS-GOMEZ, S.—PENG, I. B.—LAURE, E.—PLEITER, D.—DE WITT, S.: SAGE: Percipient Storage for Exascale Data Centric Computing. *Parallel Computing*, Vol. 83, 2019, pp. 22–33, doi: 10.1016/j.parco.2018.03.002.
 - [17] PL-Grid Consortium. The PL-Grid National Computing Infrastructure. 2020, accessed Oct 7, 2020, available at: <http://www.plgrid.pl/en>.
 - [18] REED, D. A.—DONGARRA, J.: Exascale Computing and Big Data. *Communications of the ACM*, Vol. 58, 2015, No. 7, pp. 56–68, doi: 10.1145/2699414.
 - [19] TOP500.org. The TOP500 List (November 2017). 2017, accessed Sep 14, 2020, available at: <https://www.top500.org/lists/top500/list/2017/11/>.
 - [20] TOP500.org. The TOP500 List (June 2020). 2020, accessed Sep 14, 2020, available at: <https://www.top500.org/lists/top500/list/2020/06/>.
 - [21] Triad National Security, LLC. Charliecloud. 2020, accessed Oct 7, 2020, available at: <https://hpc.github.io/charliecloud/>.
 - [22] US Department of Energy, Office of Science. Advanced Scientific Computing Research (ASCR): Scientific Grand Challenges Workshop Series. Accessed Oct 26, 2020, available at: <https://science.osti.gov/ascr/Community-Resources/Workshops-and-Conferences/Grand-Challenges>.
 - [23] CUSHING, R.—VALKERING, O.—BELLOUM, A.—SOULEY, M.—BOBAK, M.—HABALA, O.—TRAN, V.—GRAZIANI, M.—MÜLLER, H.: Process Data Infrastructure and Data Services. *Computing and Informatics*, Vol. 39, 2020, No. 4, pp. 724–756, doi: 10.31577/cai.2020.4.724.
 - [24] BOBÁK, M.—HLUCHÝ, L.—HABALA, O.—TRAN, V.—CUSHING, R.—VALKERING, O.—BELLOUM, A.—GRAZIANI, M.—MÜLLER, H.—MADOUGOU, S.—MAASSEN, J.: Reference Exascale Architecture (Extended Version). *Computing and Informatics*, Vol. 39, 2020, No. 4, pp. 644–677, doi: 10.31577/cai.2020.4.644.



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



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



Jan KAPALA received his B.Sc. degree in computer science from AGH University of Technology, Krakow, Poland in 2020 and now he is pursuing the M.Sc. degree in the M.Sc. programme Computer Science and Intelligent Systems: Artificial Intelligence and Data Analysis. Both his B.Sc. thesis and ongoing M.Sc. thesis are focused on reinforcement learning agents. He is Software Engineer at Academic Computer Centre CYFRONET AGH. His main interest is artificial intelligence.



Patryk WOJTOWICZ received his B.Sc. degree in computer science from AGH University of Technology, Krakow, Poland in 2020 and now he is pursuing the M.Sc. degree. His B.Sc. thesis and ongoing M.Sc. thesis are both focused on intelligent reinforcement learning agents. He is involved in development of the interactive execution environment platform in the PROCESS project at Academic Computer Centre CYFRONET AGH. His deep interests are artificial intelligence and its ethical aspects, and data science.



Marian BUBAK obtained his M.Sc. in technical physics and his Ph.D. in computer science from the AGH University of Science and Technology, Krakow, Poland. He is the Scientific Affairs Director and President of the Management Board of the Sano – Centre for Computational Personalised Medicine – International Research Foundation (<https://sano.science/>). He also leads the Laboratory of Information Methods in Medicine at ACC Cyfronet AGH, he is a staff member of the Department of Computer Science AGH, and the Professor of Distributed System Engineering (emeritus) at the Institute of Informatics of the

University of Amsterdam. His research interests include parallel and distributed computing and quantum computing. He served key roles in about 15 EU-funded projects and authored about 230 papers. He is a member of editorial boards of FGCS, Bio-Algorithms and Med-Systems, and Computer Science Journal.



Viet TRAN is Senior Researcher of the Institute of Informatics, Slovak Academy of Sciences (IISAS). His primary research fields are complex distributed information processing, grid and cloud computing, system deployment and security. He received 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 on preparations and solving a number of EU IST RTD 4th, 5th, 6th, 7th FP and EU H2020 projects such as PROCESS, DEEP-HybridDataCloud, EOSC-Hub and EOSC-Synergy. He is the

author or co-author of over 100 scientific publications.



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



Maximilian HÖB is Associate Scientist in the Munich Network Management Team at Ludwig-Maximilians-University Munich and Co-Coordinator of the PROCESS project, in which he also contributes to two Use Cases in the area of data management and agricultural simulation based on the Copernicus data sets. His research focuses on large scale system architectures and performance-aware containerization of HPC applications.