E-Infrastructures
H2020- INFRAEDI-2018-2020

INFRAEDI-01-2018: Pan-European High Performance Computing infrastructure and services (PRACE)

PRACE-6IP

PRACE Sixth Implementation Phase Project

Grant Agreement Number: INFRAEDI-823767

D6.4

Final report on new prototypal services
Final

Version: 1.0
Author(s): Agnes Ansari (IDRIS), Abdulrahman Azab (UiO Sigma2), Simone Bna (CINECA), Miroslaw Kupcik (PSNC), Janez Povh (UL)
Date: 20.12.2021

Project and Deliverable Information Sheet

PRACE Project	**Project Ref. №: INFRAEDI-823767**	
	Project Title: PRACE Sixth Implementation Phase Project	
	Project Web Site: https://www.prace-ri.eu/about/ip-projects/	
	Deliverable ID: < D6.4>	
	Deliverable Nature: < Report >	
	Dissemination Level: PU*	**Contractual Date of Delivery:** 31 / 12 / 2021
		Actual Date of Delivery: 21 / 12 / 2021
	EC Project Officer: Leonardo Flores Añover	

* - The dissemination level are indicated as follows: **PU** – Public, **CO** – Confidential, only for members of the consortium (including the Commission Services) **CL** – Classified, as referred to in Commission Decision 2005/444/EC.

Document Control Sheet

Document	**Title: Final report on new prototypal services**	
	ID: D6.4	
	Version: 1.0	**Status:** ***Final***
	Available at: https://www.prace-ri.eu/about/ip-projects/	
	Software Tool: Microsoft Word 2016	
	File(s): D6.4-v1.0.docx	
Authorship	**Written by:**	Agnes Ansari (IDRIS), Abdulrahman Azab (UiO Sigma2), Simone Bna (CINECA), Miroslaw Kupcik (PSNC), Janez Povh (UL)
	Contributors:	Josephine BEECH-BRANDT (EPCC), Kyriakos Ginis (GRNET), Caspar van Leeuwen (SURFSARA), Spyro NITA (EPCC), Marco RORRO (CINECA)
	Reviewed by:	Stelios Erotokritou, CaSToRC Veronica Teodor, JUELICH
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	01/10/2021	Template	Sent to contributors
0.2	05/11/2021	Draft 1	Sent back to conttributors
0.3	22/11/2021	Draft 2	Sent to reviewers
0.4	06/12/2021	Draft 3	Sent back to contributors for revision
0.5	09/12/2021	Draft 4	Revision submitted
0.7	14/12/2021	Draft 5	Second revision submitted

0.8	15/12/2021	Draft 6	Third revision submitted
1.0	20/12/2021	Final version	

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure, New prototypal services, Urgent computing, In-situ visualization, Containers, Data services

Table of Content

Project and Deliverable Information Sheet i
Document Control Sheet i
Document Status Sheet i
Document Keywords iii
Table of Content iv
List of Figures v
List of Tables v
References and Applicable Documents vi
List of Acronyms and Abbreviations xii
List of Project Partner Acronyms xiii
Executive Summary 1
1 Introduction 1
2 Service 1: Urgent Computing 2
2.1 Description of the service 2
2.2 Results achieved during the project 4
2.3 KPI 7
2.4 Future of this service 7
3 Service 2: in-situ visualisation 8
3.1 Description of the service 8
3.2 Results achieved during the project 9
3.3 Future of this service 22
4 Service 3: The deployment of containers and fully virtualised tools into HPC infrastructures 23
4.1 Description of the service 23
4.2 Results achieved during the project 29
4.3 Description of experiments and results 35
4.4 Dissemination and training activities 41
4.5 Future of this service 41
5 Service 4: Data Analytics 43
5.1 Description of the service 43
5.2 Results achieved during the project 43
5.3 Future of this service 52
6 Conclusions 53
7 Annex to Service 4 – Data Analytics 54
7.1 Survey form 2021 54

7.2 Additional survey results 59
7.3 Secure architecture Operation 60
7.4 Installation instructions 61
7.5 User manuals 78

List of Figures

Figure 1: Instantaneous iso-surface of the lambda2 criterion coloured with the x-component of the velocity 11
Figure 2: Q-criterion (left) and 2D-streamlines (right) visualisation pipelines 12
Figure 3: A visualiSation from full in-situ visualisation pipeline 14
Figure 4: Box and whisker plots of strong scaling for 58M model, single CSV output (top left), double CSV output (top right), XML output(bottom left). All runs are using 32 processes per node and 3 threads per process. Small coloured points are outliers. 19
Figure 5: Melissa Architecture Overview 20
Figure 6: Results of the OSU All-to-all benchmark for the native MPI and MPICH 3.1.4 container. The MPI in the container is replaced at runtime by the native MPICH MPI hook used by Sarus. 38
Figure 7: Comparison of wall clock execution time, performance, and speedup between native and Sarus-deployed container versions of GROMACS on Piz Daint 40
Figure 8: Distribution of answers by scientific domain 44
Figure 9: Software stacks, ML/DL frameworks and libraries 45
Figure 10 : Set of graphical tools used by Data Analytics users 45
Figure 11: Dataset origin 46
Figure 12: Container usage for Data Analytics 46
Figure 13: Secure architecture to use Jupyter Notebooks and Tensorboard 47
Figure 14: JupyterHub setup on a SLURM cluster at SURF. 48
Figure 15: The four stages of AI Development supported by Acumos 50
Figure 16 : Users access to the PRACE infrastructure 59
Figure 17 : Project sizes 59
Figure 18 : Dataset type 60
Figure 19 : The Spark architecture 84

List of Tables

Table 1 MIGALE - T3L: in-situ visualisation relative cost with respect to time spent for advancing the solution of a time step without in-situ [Tpipeline/Ttimestep] with the variation of the number of degrees of freedom [N_{dof}] and the number of sub-elements [n_{se}] 11
Table 2: OpenFOAM – pump impeller: in-situ overhead of the Q-criterion and 2D-streamline pipelines respect to the elapsed time of ten iterations of the PIMPLE algorithm 13
Table 3: STREAmS use-cases parameters: Reynolds number, number of GPUs, non-dimensionaltime-step, iteration elapsed time 14
Table 4: Elapsedd times [s] for in-situ visualisation pipeline sections: slice filter, contour filter, additional times (data preparation and image rendering). For contour filter VTKm is also considered 15
Table 5: Results of the intra-node scalability testing 16

Table 6: Results of the inter-node scalability testing 17
Table 7: HermiTux and HermitCore are the only unikernels that meet our criteria 29
Table 8: Problems solved by the Bots benchmarks, and their respective implementations. 34
Table 9: gigaflops per second performance attained by the two applications 36

References and Applicable Documents

[1] Kupczyk, M., Kaliszan, D., Stoffers, H., Wilson, N., Moll, F., "Urgent Computing service in the PRACE Research Infrastructure Witepaper," PRACE-4IP, 2017.

[2] Leong, Siew Hoon; Frank, Anton; Kranzlmüller, Dieter, "Leveraging e-Infrastructures for Urgent Computing," *Procedia Computer Science,* vol. 18, pp. 2177-2186, 2013.

[3] Kurowski, Krzysztof; Oleksiak, Ariel; Piatek, Wojciech; Węglarz, Jan, "Impact of urgent computing on resource management policies, schedules and resources utilization," *Procedia Computer Science,* pp. 1713-1722, 9 2012.

[4] Baliś, B., Bubak, M., Herezlak, D., Nowakowski, P., Pawlik, M., Wilk, B., "Smart levee monitoring and flood decision support system: reference architecture and urgent computing management," *Procedia Computer Science 108C*, pp. 2220-2229, 2017.

[5] Marru, Suresh, Gannon, Dennis, Nadella, Suman, Beckman, Pete, Weber, Daniel, Brewster, Keith, Droegemeier, Kelvin, "LEAD Cyberinfrastructure to Track Real-Time Storms Using SPRUCE Urgent Computing," *CTWatch Quarterly,* no. 4, 1 3 2008.

[6] "Swarm mode overview," Docker Inc, 2020. [Online]. Available: https://docs.docker.com/engine/swarm/.

[7] C. eFlows4HPC, "Enabling dynamic and Intelligent workflows in the future EuroHPC ecosystem," [Online]. Available: https://eflows4hpc.eu.

[8] Moss, Sebastian, "TACC to expand Frontera supercomputer for urgent computing workloads," DCD, 2020.

[9] Group, EDANYA Research, "HySEA codes," Malaga Univ, 2021. [Online]. Available: https://edanya.uma.es/hysea/index.php.

[10] EDANYA, "Differential Equations, Numerical Analysis and Applications Research Group," Universidad de Málaga, [Online]. Available: https://www.uma.es/edanya.

[11] Mondaic, "SALVUS," Mondaic AG/Ltd. ETH spinoff, [Online]. Available: https://mondaic.com/.

[12] SLURM, "scontrol - view or modify Slurm configuration and state.," SLURM, [Online]. Available: https://slurm.schedmd.com/scontrol.html.

[13] NVIDIA, "vGPU Migration Support," NVIDIA, [Online]. Available: https://docs.nvidia.com/grid/latest/grid-vgpu-release-notes-vmware-vsphere/index.html#vgpu-migration-support.

[14] United-Nations-Office, "2020: the non-COVID Year in Disasters - Global Trends and Perspectives," United Nations Office for Disaster Risk Reduction, [Online]. Available: https://www.undrr.org/publication/2020-non-covid-year-disasters.

[15] curlytales, "10 Biggest Natural Disasters Of 2020 That Shook The World Costing Money & Lives," curlytales, [Online]. Available: https://curlytales.com/9-natural-disasters-that-have-already-happened-in-just-5-months-of-2020/.

[16] UNDRR, "Record hurricane season and major wildfires – The natural hazard figures for 2020," [Online]. Available: https://www.preventionweb.net/news/record-hurricane-season-and-major-wildfires-natural-hazard-figures-2020.

[17] "NOAA National Centers for Environmental Information (NCEI) U.S. Billion-Dollar Weather and Climate Disasters," NCEI, 2021.

[18] https://www.top500.org/.

[19] Bauer, Andrew et. al., "In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms," *Computer Graphic Forum*, vol. 35, pp. 577-597, 2016.

[20] "Kitware: ParaView," (2002–2016). [Online]. Available: http://www.paraview.org, .

[21] Ayachit, U., et al., "Ayachit, U., et al.: Paraview catalyst: enabling in situ data analysis and visualization," *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, vol. ACM, p. 25–29, 2015.

[22] Bassi, Francesco & Botti, L. & Colombo, A. & Crivellini, Andrea & Ghidoni, Antonio & Nigro, Alessandra & Rebay, "Time Integration in the Discontinuous Galerkin Code MIGALE," *Unsteady Problems*, pp. 3-10, 2015.

[23] [Online]. Available: https://www.openfoam.com/.

[24] [Online]. Available: https://github.com/matteobernardini/STREAmS.

[25] Franciolini, M., L. Botti, A. Colombo, A. Crivellini, "P-multigrid matrix-free discontinuous Galerkin solution strategies for the under-resolved simulation of incompressible turbulent flows," *Computers and Fluids*, 2020.

[26] Bassi, F. et al., "F. Bassi, L. Botti, A. Colombo, A. Crivellini, M. FranciolA p-adaptive matrix-free Discontinuous Galerkin method for the Implicit LES of incompressible transitional flows," *Flow, Turbulence and Combustion*, vol. 105, no. 2, pp. 437-470, 2020.

[27] Coupland, C., D. Brierley, "Transition in turbomachinery flows. Final report, BRITE/EURAM Project AERO-CT92-0050," 1996.

[28] Pedersen, N., Larsen, P. S., and Jacobsen, C. B. , "Flow in a Centrifugal Pump Impeller at Design and Off-Design Conditions—Part I: Particle Image Velocimetry (PIV) and Laser Doppler Velocimetry (LDV) Measurements," *Journal of Fluids Engineering*, vol. 12, 2003.

[29] Kok, J., Dol, H., Oskam, B., and van der Ven, H., "Extra-large eddy simulation of massively separated flows," *Aerospace Sciences Meeting and Exhibit.*, 2004.

[30] Yoshizawa, A., and Horiuti, K., "A statistically-derived subgrid-scale kinetic energy model for the large-eddy simulation of turbulent flows," *Journal of the Physical Society of Japan*, vol. 54, no. 8, p. 2834–2839, 1985.

[31] Dupont, C. Haddad, J. Debiéve, "Space and time organization in a shock-induced separated boundary layer," *J. Fluid Mech*, vol. 559, p. 255–277, 2006.

[32] Pirozzoli, S., M. Bernardini, F. Grasso, "On the dynamical relevance of coherent vortical structures in turbulent boundary layers," *J. Fluid Mech*, vol. 648, p. 325–349, 2010.

[33] [Online]. Available: https://sylabs.io/singularity/.

[34] [Online]. Available: https://spack.io/.

[35] [Online]. Available: https://project.inria.fr/damaris/.

[36] [Online]. Available: https://www.code-saturne.org/cms/web/.

[37] Joshua Bowden, Francois Tessier, Charles Deltel, Simone Bnà, Gabriel Antoniu, "In-situ visualization using Damaris: the Code Saturne use case. PRACE: Partnership for Advanced Computing in Europe. 2021," PRACE White Papers, 2021.

[38] [Online]. Available: https://gitlab.inria.fr/melissa.

[39] [Online]. Available: https://spack.io/.

[40] "E4S," [Online]. Available: https://e4s-project.github.io/.

[41] "CSCS," [Online]. Available: https://www.cscs.ch/computers/piz-daint/.

[42] "Developer," [Online]. Available: https://developer.nvidia.com/cuda-code-samples.

[43] "Developer," [Online]. Available: https://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/nbody/doc/nbody_gems3_ch31.pdf.

[44] "Hub," [Online]. Available: https://hub.docker.com/r/ethcscs/cudasamples/tags.

[45] "Mvapich," [Online]. Available: http://mvapich.cse.ohio-state.edu/benchmarks/.

[46] "MPICH," [Online]. Available: https://www.mpich.org/abi/.

[47] "Sarus," [Online]. Available: https://sarus.readthedocs.io/en/1.3.3/_downloads/f11cdd5df75fb08b21b054b5861c0a20/Dockerfile.ubuntu1804+cuda92+mpich314+osu.

[48] "sarus," [Online]. Available: https://sarus.readthedocs.io/en/stable/_downloads/a5fa1dbca9d4c778e819d1e8aa88fe27/all_gather.cpp.

[49] "Gromacs," [Online]. Available: http://www.gromacs.org/.

[50] "Hecbiosim," [Online]. Available: http://www.hecbiosim.ac.uk/benchmarks.

[51] "Hecbiosim," [Online]. Available: http://www.hecbiosim.ac.uk/media/jdownloads/Software/gromacs.tar.gz.

[52] "CSCS," [Online]. Available: https://user.cscs.ch/computing/applications/gromacs/.

[53] "Hub docker," [Online]. Available: https://hub.docker.com/r/ethcscs/gromacs/tags.

[54] "Tensorflow," [Online]. Available: https://www.tensorflow.org/.

[55] "Github," [Online]. Available: https://github.com/horovod/horovod.

[56] "Github," [Online]. Available: https://github.com/tensorflow/benchmarks.

[57] "Freebsdfoundation," [Online]. Available: https://freebsdfoundation.org/ .

[58] "Hub docker," [Online]. Available: https://hub.docker.com/r/ethcscs/mpich/tags.

[59] Brayford, D. and S. Vallecorsa,, "Deploying Scientific Al Networks at Petaflop Scale on Secure Large Scale HPC Production Systems with Containers," *Proceedings of the Platform for Advanced Scientific Computing Conference,* 2020.

[60] Brayford, D., Vallecorsa, S., Atanasov, A., Baruffa F. and W. Riviera, "Deploying AI Frameworks on Secure HPC Systems with Containers," *2019 IEEE High Performance Extreme Computing Conference (HPEC), ,* pp. 1-6, 2019.

[61] Höb M., Kranzlmüller D., "Enabling EASEY Deployment of Containerized Applications for Future HPC Systems. In: Krzhizhanovskaya V. et al. (eds) Computational Science," *Lecture Notes in Computer Science,* vol. 12137, 2020.

[62] Brennan, John, Momme Allalen, David Brayford, Kenneth Hanley, Luigi Iapichino, Lee J. O'Riordan and Niall Moran, "Deploying Containerized QuantEx Quantum Simulation Software on HPC Systems," *CANOPIE SC21 Workshop.*

[63] Marocchi , "Evaluation and Benchmarking of Singularity MPI Containers on EU Research e-Infrastructures," *PROCEEDINGS OF CANOPIE-HPC* , 2019.

[64] Muscianisi, G. Fiameni,G., Azab A., "Singularity GPU containers execution on HPC Cluster," *Springer Nature Switzerland,* pp. 61-68, 2019.

[65] Azab, A., Muscianisi, G., Wiber, G. Fernandez, C., "Evaluation of Linux Container and full virtualization for HPC Applications in PRACE 5IP, PRACE 5IP Task 6.2.5," *Whitepaper, 2019 (https://prace-ri.eu/wp-content/uploads/WP286.pdf).*.

[66] "PRACE," [Online]. Available: http://www.prace-ri.eu.

[67] "Tensorflow," [Online]. Available: https://www.tensorflow.org/.

[68] "Hub docker," [Online]. Available: https://hub.docker.com/r/ethcscs/horovod/tags.

[69] "Arxiv," [Online]. Available: https://arxiv.org/abs/1512.03385.

[70] Kurtzer GM, Sochat V, Bauer MW, "Singularity: Scientific containers for mobility of compute. PLoS ONE," no. https://doi.org/10.1371/journal.pone.0177459, 2017.

[71] [Online]. Available: /opt/hermit/lib/gcc/x86_64-hermit/6.3.0/include/memory.h.

[72] [Online]. Available: to/opt/hermit/x86_64-hermit/include/hermit/memory.h. .

[73] Merkel, D., " Docker: lightweight linux containers for consistent development and deployment.," *Linux Journal, 2014(239), 2.,* 2014.

[74] Kivity, Avi, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti, and Vlad Zolotarov.Osv, "Optimizing the operating system for virtual machines.," *USENIX Annual Technical Con-ference (USENIX ATC 14), Philadelphia, PA, June,* pp. 61-72, 2014.

[75] "Mvapich," [Online]. Available: https://www.mpich.org/abi/.

[76] "Containerday," [Online]. Available: https://2019.containerday.it/.

[77] "Eventi Cineca," [Online]. Available: https://eventi.cineca.it/en/hpc/containerization-hpc .

[78] "PRACE," [Online]. Available: https://events.prace-ri.eu/event/1149/ .

[79] "PRACE," [Online]. Available: https://events.prace-ri.eu/event/1229/.

[80] [Online]. Available: https://jupyter.org/hub.

[81] [Online]. Available: https://github.com/jupyterhub/batchspawner.

[82] [Online]. Available: https://github.com/jupyterhub/wrapspawner.

[83] [Online]. Available: https://www.mlflow.org.

[84] [Online]. Available: https://polyaxon.com.

[85] [Online]. Available: https://www.acumos.org.

[86] [Online]. Available: https://github.com/open-ce.

[87] [Online]. Available: https://osuosl.org/services/powerdev/opence.

[88] [Online]. Available: https://github.com/open-ce/open-ce-builder.

[89] "Anaconda," [Online]. Available: https://www.anaconda.com/products/individual.

[90] "Keras," [Online]. Available: https://keras.io/.

[91] "Scikit," [Online]. Available: https://scikit-learn.org/stable/.

[92] "Jupyter," [Online]. Available: https://jupyter.org/.

[93] "Scikit," [Online]. Available: https://scikit-learn.org/stable/.

[94] "Nvidia," [Online]. Available: https://developer.nvidia.com/nvidia-tensorrt-download.

[95] "Spark," [Online]. Available: https://spark.apache.org/docs/latest/index.html.

[96] "Spark," [Online]. Available: https://spark.apache.org/docs/2.4.0/spark-standalone.html.

[97] "Spark," [Online]. Available: https://spark.apache.org/docs/latest/submitting-applications.html.

[98] "Cirrus," [Online]. Available: https://www.cirrus.ac.uk/ .

[99] PRACE 5-IP [Online]. Available: https://prace-ri.eu/wp-content/uploads/5IP-D6.4.pdf

List of Acronyms and Abbreviations

aisbl	Association International Sans But Lucratif (legal form of the PRACE-RI)
API	Application Programming Interface
BDF	Backward Differentiation Formula scheme
BSD	Berkeley Software Distribution
CFD	Computational Fluid Dinamics
CoE	Center of Excellence
EoCoE-II	Energy Oriented Center of Excellence: toward exascale for energy
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture (NVIDIA)
ESSI	European Environment for Scientific Software Installation
FTRT	Faster Than Real Time
GB	Giga (= 2^{30} ~ 10^9) Bytes (= 8 bits), also GByte
Gb/s	Giga (= 10^9) bits per second, also Gbit/s
GB/s	Giga (= 10^9) Bytes (= 8 bits) per second, also GByte/s
GÉANT	Collaboration between National Research and Education Networks to build a multi-gigabit pan-European network. The current EC-funded project as of 2015 is GN4.
GFlop/s	Giga (= 10^9) Floating point operations (usually in 64-bit, i.e. DP) per second, also GF/s
GHz	Giga (= 10^9) Hertz, frequency =10^9 periods or clock cycles per second
GPU	Graphic Processing Unit
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing
HPL	High Performance LINPACK
KB	Kilo (= 2^{10} ~10^3) Bytes (= 8 bits), also KByte
KPI	Key Performance Indicator
KPTI	Kernel page-table isolation
LINPACK	Software library for Linear Algebra
LXD	Linux Container Daemon
MFlop/s	Mega (= 10^6) Floating point operations (usually in 64-bit, i.e. DP) per second, also MF/s
MRF	Multiple Reference Frame
NCEI	National Centers for Environmental Information
PA	Preparatory Access (to PRACE resources)
PDAF	Parallel Data Assimilation Framework
PRACE	Partnership for Advanced Computing in Europe; Project Acronym
PRACE 2	The upcoming next phase of the PRACE Research Infrastructure following the initial five year period.
RANS	Reynolds-averaged Navier-Stokes
RI	Research Infrastructure
SLA	Service Level Agreement
SST	Shear Stress Transport Turbulence model
TB	Tera (= 2^{40} ~ 10^{12}) Bytes (= 8 bits), also TByte
TEWS	Tsunami Early Warning Systems
TFlop/s	Tera (= 10^{12}) Floating-point operations (usually in 64-bit, i.e. DP) per second, also TF/s
TLB	Translation Lookaside Buffer
TNT	Turbulent/Non-Turbulent

Tier-0	Denotes the apex of a conceptual pyramid of HPC systems. In this context the Supercomputing Research Infrastructure would host the Tier-0 systems; national or topical HPC centres would constitute Tier-1
UC	Urgent Computing
UEABS	Unified European Applications Benchmark Suite
X-LES	eXtra Large Eddy Simulation

List of Project Partner Acronyms

BADW-LRZ	Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften, Germany (3rd Party to GCS)
BILKENT	Bilkent University, Turkey (3rd Party to UHEM)
BSC	Barcelona Supercomputing Center - Centro Nacional de Supercomputacion, Spain
CaSToRC	The Computation-based Science and Technology Research Center (CaSToRC), The Cyprus Institute, Cyprus
CCSAS	Computing Centre of the Slovak Academy of Sciences, Slovakia
CEA	Commissariat à l'Energie Atomique et aux Energies Alternatives, France (3rd Party to GENCI)
CENAERO	Centre de Recherche en Aéronautique ASBL, Belgium (3rd Party to UANTWERPEN)
CESGA	Fundacion Publica Gallega Centro Tecnológico de Supercomputación de Galicia, Spain, (3rd Party to BSC)
CINECA	CINECA Consorzio Interuniversitario, Italy
CINES	Centre Informatique National de l'Enseignement Supérieur, France (3 rd Party to GENCI)
CNRS	Centre National de la Recherche Scientifique, France (3 rd Party to GENCI)
CSC	CSC Scientific Computing Ltd., Finland
CSIC	Spanish Council for Scientific Research (3rd Party to BSC)
CYFRONET	Academic Computing Centre CYFRONET AGH, Poland (3rd Party to PNSC)
DTU	Technical University of Denmark (3rd Party of UCPH)
EPCC	EPCC at The University of Edinburgh, UK
EUDAT	EUDAT OY
ETH Zurich (CSCS)	Eidgenössische Technische Hochschule Zürich – CSCS, Switzerland
GCS	Gauss Centre for Supercomputing e.V., Germany
GÉANT	GÉANT Vereniging
GENCI	Grand Equipement National de Calcul Intensif, France
GRNET	National Infrastructures for Research and Technology, Greece
ICREA	Catalan Institution for Research and Advanced Studies (3rd Party to BSC)
INRIA	Institut National de Recherche en Informatique et Automatique, France (3rd Party to GENCI)
IST-ID	Instituto Superior Técnico for Research and Development, Portugal (3rd Party to UC-LCA)
IT4I	Vysoka Skola Banska - Technicka Univerzita Ostrava, Czech Republic
IUCC	Machba - Inter University Computation Centre, Israel
JUELICH	Forschungszentrum Jülich GmbH, Germany
KIFÜ (NIIFI)	Governmental Information Technology Development Agency, Hungary
KTH	Royal Institute of Technology, Sweden (3rd Party to SNIC-UU)

KULEUVEN	Katholieke Universiteit Leuven, Belgium (3rd Party to UANTWERPEN)
LiU	Linkoping University, Sweden (3rd Party to SNIC-UU)
MPCDF	Max Planck Gesellschaft zur Förderung der Wissenschaften e.V., Germany (3rd Party to GCS)NCSA NATIONAL CENTRE FOR SUPERCOMPUTING APPLICATIONS, Bulgaria
NTNU	The Norwegian University of Science and Technology, Norway (3rd Party to SIGMA2)
NUI-Galway	National University of Ireland Galway, Ireland
PRACE	Partnership for Advanced Computing in Europe aisbl, Belgium
PSNC	Poznan Supercomputing and Networking Center, Poland
SDU	University of Southern Denmark (3rd Party to UCPH)
SIGMA2	UNINETT Sigma2 AS, NorwaySNIC-UU Uppsala Universitet, Sweden
STFC	Science and Technology Facilities Council, UK (3rd Party to UEDIN)
SURF	SURF is the collaborative organisation for ICT in Dutch education and research
TASK	Politechnika Gdańska (3rd Party to PNSC)TU WienTechnische Universität Wien, Austria
UANTWERPEN	Universiteit Antwerpen, Belgium
UC-LCA	Universidade de Coimbra, Labotatório de Computação Avançada, Portugal
UCPH	Københavns Universitet, Denmark
UEDIN	The University of Edinburgh
UHEM	Istanbul Technical University, Ayazaga Campus, Turkey
UIBK	Universität Innsbruck, Austria (3rd Party to TU Wien)
UiO	University of Oslo, Norway (3rd Party to SIGMA2)
UL	UNIVERZA V LJUBLJANI, Slovenia
ULIEGE	Université de Liège; Belgium (3rd Party to UANTWERPEN)
U Luxembourg	University of Luxembourg
UM	Universidade do Minho, Portugal, (3rd Party to UC-LCA)
UmU	Umea University, Sweden (3rd Party to SNIC-UU)
UnivEvora	Universidade de Évora, Portugal (3rd Party to UC-LCA)
UnivPorto	Universidade do Porto, Portugal (3rd Party to UC-LCA)
UPC	Universitat Politècnica de Catalunya, Spain (3rd Party to BSC)
USTUTT-HLRS	Universitaet Stuttgart – HLRS, Germany (3rd Party to GCS)
WCSS	Politechnika Wroclawska, Poland (3rd Party to PNSC)

Executive Summary

In this deliverable, we present results obtained in Task 6.2 of Work Package 6 of the PRACE-6IP project. This task focused on four new services that had the potential to address some of the widely recognised needs in scientific computing and were also related with efforts toward the exascale transition. These services are: urgent computing, in-situ visualisation, the deployment of containers and full virtualised tools into HPC, and data analytics. All of them were already investigated within the preceding PRACE-5IP project and were smoothly continued within this project.

The main objective of this deliverable is therefore to present the work done within each service. The deliverable has four main sections – one for each service, which contains the main results obtained within each service. These results are: (i) pilot implementation of two scientific codes in urgent computing mode and incorporating the follow-up activities into a new EU project eFlows4HPC, (ii) succesfull installation of three in-situ frameworks (Catalyst, Damaris, Melissa) into seven PRACE HPC clusters and instrumentation of several CFD codes to be coupled with in-situ frameworks, (iii) several trainings and dissemination activities and support for 23 use cases where containers combined with parallelisation were a key-enabling technology, and (iv) automatic installation instructions and user manuals for a list of nine data services were created. The deliverable also provides links to the other results of the Task 6.2, like the scientific papers and white papers containing results obtained during during this task.

Each section contains a description of each specific service, the list of planned activities, report of the work done within PRACE-6IP, final conclusion based on the results of the activities and a proposal on how to continue with the service. An Annex contains important results developed within the Data analytics service, which are expected to be appreciated by scientific communities related to Big Data analysis.

This document can also act as a basis for preparation of the next PRACE implementation phase project, if there will be an opportonuty to prepare it.

1 Introduction

An efficient and state-of-the-art pre-exascale HPC infrastructure at a European level should be ready to operate innovative services to address scientific, technological and societal challenges. In Task 6.2 of the PRACE-6IP project, we have examined four such services. The main objective of this deliverable is to report on results, obtained during this task.

Following the interest of project partners, four groups of partners – one for each new service – were defined. The groups were coordinated by ULFME, which was the Task 6.2 coordinator. They were composed of:

- Service 1: **Urgent Computing (UC)**: the coordinator of this service and the only member of the working group was PSNC;
- Service 2: **In-situ visualisation**: the coordinator of this service was CINECA and other partners involved in were INRIA and HLRS;
- Service 3: **The deployment of containers and full virtualised tools into HPC infrastructures**: work on this service was coordinated by UiO and the other contributing partners were INRIA and CEA;

- Service 4: **Data Analytics**: this service was coordinated by IDRIS-CNRS. The other partners involved in this service were CINECA, SURF, GRNET, NCSA, UL and EPCC.

During our work, in the first 3 months, we initially developed operational plans for each service and then we followed these plans until the end of October 2021, when the main components of this deliverable were written. For each service, this deliverable contains the:

(i) service description
(ii) description of the work carried out within the PRACE-6IP project on the service
(iii) proposition on how to continue with this service in the period beyond PRACE-6IP project.

For each service we have also developed one Key Performance Indicator (KPI) which encapsulated the main ingredients of the service. These KPIs are reported at the end of each service description.

During the finalisation of Task 6.2, we have collected important technical details in the form of several white papers and scientific papers. The main outcomes of the four services are appended to this deliverable as an Annex. The deliverable will be an useful reading for technicians that work on implementations of different software stacks on an HPC and for researchers who want to use these stacks within their research projects and programs.

2 Service 1: Urgent Computing

2.1 Description of the service

Urgent Computing (UC) enables responsible bodies to make conscious decisions by supporting computations of simulated predictions of time critical events, usually related to natural hazards. Unfortunately, most domains of science cannot afford dedicated resources for their urgent computing problems.

As PRACE is an HPC infrastructure designed for research, the compute-only part of most of its supercomputers is not fault-tolerant against several sorts of hardware failure and certainly not against power failures. Since redundancy and fault tolerance is not for free, and hardware and power failures occur quite infrequently, it is more cost-efficient to spend money on total capacity and to simply reschedule and rerun the jobs that have suffered from an occasional failure than to spend more money on resilience and redundancy.

On PRACE systems it is also customary that maintenance intervals are planned and announced in advance, without giving users a back-up production site during maintenance. If the announcements are well in advance, this is no problem for regular research jobs. For UC jobs the matter is quite different. It is a defining hallmark for the context of UC that the demand is only foreseeable for a short period prior to execution. Means to synchronise, by delaying or speeding up the course of certain events, may be lacking altogether. The urgency usually implies that there is no second chance. Starting a job after the deadline has past or having to rerun a failed job will make the result, although correct, useless simply because it is too late [1].

Within PRACE, we are not able to propose a uniform policy of urgent scenario deployment on PRACE sites. PRACE sites (Tier-0 and Tier-1) have got their own, independent machine

utilisation scenario and, according to the analysis – currently no PRACE site is eligible to run UC job without thorough reconfiguration of the local job management system reconfiguration. It is worth mentioning, that three PRACE sites were actively involved in UC scenarios evaluation in the past: LRZ [2], PSNC [3], and Cyfronet [4]. The following responses to an urgent computing request are possible in general:

1. Scheduling the urgent job as "next-to-run" in a priority queue. This approach is simple and highly recommended as a possible response for all resource providers. No running computation is killed; the impact on normal use is low. The urgent job will begin when all of the running jobs complete for a given set of CPUs. Unfortunately, this wait could go up to hours or even days. It is not suitable for UC mode.
2. Suspending running jobs and immediately launching the urgent job. This will then force some memory paging, but the suspended job could be resumed later. Node crashes and failed network connections can be an obstacle in reviving suspended jobs. The benefit of this policy is that urgent jobs will begin almost immediately, making this option attractive in some cases. This scenario was analysed in PRACE-6IP with the support of two important real applications.
3. Forcing a checkpoint/restart of running jobs and re-queuing the urgent job as the next to run. This response is similar to the previous response but safely moves the checkpoint to a location where it can then be used to restart on alternative resources. Architectures supporting system-based checkpoint/restart can be used to support urgent computing where reliable. This checkpointing for large-memory systems could take 30 minutes or more depending on I/O and storage rates. Also, checkpoint/restart feature might be implemented in the application to ensure system independency.
4. Killing all running jobs and queuing the urgent job as next to run. Clearly this response is drastic and frustrating to the users who will lose their computation. Nevertheless, it will ensure that extremely urgent computations begin immediately after running jobs are killed [5]. The policy is technically easy to implement, but it spoils the ordinary users' trust and definitely decreases an HPC center's reputation.
5. Novel architecture exploitation: cloud processing, e.g. Docker SWARM [6]. Docker swarm is a container orchestration tool, meaning that it allows the user to manage multiple containers deployed across multiple host machines. One of the key benefits associated with the operation of a docker swarm is the high level of availability offered for applications and a very dynamic environment. The application must be aware of that to exploit all pros. This scenario has not been evaluated in PRACE-6IP in the context of Urgent Computing due to the lack of native cloud-based infrastructure. The deployment of such technology is a subject to analysis, how the resources shall be managed in the future.

Any operational platform supporting UC case implies that the environment is in a state of "warm standby" for the real event. To be, and remain, in a state of warm standby:

- All needed application software must be pre-installed;
- One or more data sets suitable for validation must be pre-installed;
- There must be a validation protocol using pre-installed software and pre-installed data;
- Runs to execute the validation protocol must be scheduled regularly to verify that the applications keep performing as they should, especially after system changes, software upgrades on general purpose and computational libraries, etc.;

- Validation runs can be regular batch jobs. A budget of sufficient core hours to perform these jobs regularly must be allocated;
- Since pre-installed software and data may be damaged by human error, or hardware malfunctioning, or even the above noted updating of other software components, there should also be a regularly tested procedure to quickly restore – reinstall, relink, recompile, reconfigure, whatever applies – the pre-installed components;
- While software validation can fit into the regular production environment, the workflow of a real event necessarily requires more;
- Platforms supporting a UC case must have a mechanism to raise the emergency flag, which can be executed by a preselected class of users: UC users, UC operators;
- Raising of the emergency flag must enable the availability of the required compute and storage resources in due time;
- Although it is project dependent, the exact amount of required compute and storage resources, and the exact due time should be agreed in advance by the involved parties supplying and using the platform;
- How the freeing of adequate resources in due time is implemented should be at the discretion of the platform supplying side. Some sides may want to use pre-emption of already running jobs, others may e.g. have a large enough dedicated partition for jobs with a fairly short wall clocks time that they can drain from regular job usage - responses to an urgent computing request;
- Complete workflow scenarios, including the availability of resources in due time, must be regularly practiced as "dry runs" as well;
- The regular testing of workflow scenarios is potentially much more disruptive for the normal production work than the above-mentioned regular software validation and their frequency must be agreed upon in advance, at the intake of a UC project. The budget in core hours allocated for a UC project should be sufficient to cover the actual loss of economic capacity resulting from the agreed upon level of workflow scenario dry runs.

The design of the complete UC workflow is not a topic of PRACE-6IP. It will be worked out in another EU project eFlows4HPC (contract No 955558) [7]). They work on the implementation of real UC workflows for Tsunami simulation and Seismic simulation. PRACE-6IP strongly co-operated on testing HPC core kernels in order to discover the characteristics of the usage and load parameters on the selected machine.

2.2 Results achieved during the project

The goal of the PRACE-6IP project was to propose a PRACE service dedicated to running specialised applications with given time constraints. UC mode is the most eligible to adapt such scenarios. The paradigm of "on demand computing" is also interesting but it does not address the constraints as UC policy must assure. At the time, when the design of the service was starting, there was no real customer in Europe known to us. Even more, we were not aware of anyone really interested to take the advantage of the advanced features and willing to contribute. In Europe, the situation changed in 2019 approximately.

The plan assumed to find a good example of the real case eligible to run in UC scenario mode and according to the observation and analysis during the runtime – preparing the proposal of the policy for these applications. We are aware, that the UC mode is uncommon in HPC centers. There is only one evidence that the UC mode has been deployed in real environment. It is on

the Frontera system (peak performance of 23.5 Petaflops) located in TACC (US). In 2020, the machine has been extended for helping fight Covid-19 and producing emergency storm surge simulations for hurricanes making landfall in the Gulf of Mexico [8]. The part of the machine is being booked for UC runs only. This scenario points out the general decision process, how to define SLA for UC and other resources' users affected by unordered job processing.

The previous PRACE-5IP project had started piloting co-operation with CoE: Center of Excellence for Exascale in Solid Earth (ChEESE), ID: 823844, as described in the PRACE 5IP deliverable D6.4 [99]. The development of UC services should be based on the European open science and FAIR data policy with matured and operational open codes and use cases including all the data and logistics required up to the distilling of the results in forms than can be used for expert opinions and decision making.

This should involve co-design and co-development between ChEESE, PRACE-IP projects and the end users. ChEESE CoE considered "urgent" simulations as possible use cases within a testing phase to identify and assess the required services, resource management and policy access in relation with different workflow patterns and data logistics, while checking the feasibility of effectively contributing to future emergencies. ChEESE had a special focus on developing at least 10 pilots and services, but 2 of them promised well in making use of urgent supercomputing with disaster resilience purposes. The main interesting use case pilots are the following:

Faster than real-time tsunami simulations

FTRT tsunami computations are crucial in the context of Tsunami Early Warning Systems (TEWS) and in the context of post-disaster management. Greatly improved and highly efficient computational methods are the first raw ingredient to achieve extremely fast and effective calculations. HPC facilities have the role to bring this efficiency to a maximum while drastically reducing computational times. In addition, inputs from an urgent seismic simulation can be exploited for physics-based urgent tsunami simulations. A typical case of probabilistic tsunami forecasting would use from thousands to tens of thousands tsunami simulations for different realisations of the parameters describing the causative source.

Near real-time seismic scenarios

By including full 3D physical models and topography, the level of details that simulations can attain and the quantities of interest that can be inferred (e.g. shaking time, peak ground acceleration, and response spectral values at each point on the surface) are highly valuable to analyse the outcome of earthquakes with high resolution. If a solution can be attained at time spans that are sufficient for disaster management (i.e. hours), urgent computing seismic scenario simulations might become useful seismic resilience tools.

PSNC, as the only PRACE site involved in the UC service became a member of the eFlows4HPC consortium. Till that time, the co-operation with the application owners or developers was hampered due to lack of any bilateral agreement or legal tender between PRACE and another consortium. In PRACE-6IP, the analysis of the selected applications was performed. It was expected to get the answer how to organise the machine load policy, what would happen with the entire machine state when the UC job is placed immediately in the scenario, i.e., by suspending running jobs to memory and immediately launching the urgent job.

Testing environment:

EAGLE PRACE Tier-1 partition:

9 nodes: [Intel Xeon 6242, 2x16 cores, 384 GB, 8xNvidia V100 32GB]; 0.56 PFlops

OS: GNU Linux

LRMS: SLURM 20.11.8.

Application 1. Faster than real-time tsunami simulations (PTF/FTRT)

HySEA [9] (Hyperbolic Systems and Efficient Algorithms) software consists of a family of geophysical codes based on either single layer, two-layer stratified systems or multilayer shallow water models. HySEA is a high-performance software package developed by the EDANYA group at the University of Málaga, Spain [10] , for the simulation of geophysical flows, including tsunamis generated by earthquakes or landslides, river floodings, sediment transport, turbidity currents, etc.

Tsunami-HySEA is implemented in CUDA language, in double precision and in structured meshes, using two-way nested meshes techniques and for multi-GPU architectures.

Requirements

- CMake >= 3.0
- A CUDA-capable GPU
- CUDA >= 6.0
- NetCDF
- PnetCDF (Parallel NetCDF) (only for the simulator)

Before running the job, some essential files must be prepared:

- Bathymetry file
- Parameters input files
- File for managing the number of GPUs used

The experimental runs on EAGLE PRACE Tier-1 machine show that typical HySEA job requires 9 GB of local memory per 1 GPU. It is also important to make a note, that final output data set needs 3.5 TB (depends on the mesh density, number of points).

As a result of that tests is the theoretical possibility to run Tsunami-HySEA job in UC mode without thorough devastation of the machine load performance. It is possible to suspend to memory (relatively small) running jobs in order to free resources, however not all GPU jobs restarted successfully. The examination of this phenomena was not a goal of PRACE-6IP.

Application 2. Near real-time seismic scenarios (UCIS4EQ)

Salvus [11] is a software suite focusing on high-performance full waveform modelling and inversion was used for HDF5 meshes generating. It consists of five components:

- SalvusCompute: Parallel high-performance solver for spectral-element wave propagation on unstructured meshes;
- SalvusMesh: Library and toolbox to build meshes in both two and three dimensions;
- SalvusFlow: Workflow orchestration and remote job execution framework;

- SalvusOpt: Non-linear optimisation framework;
- SalvusProject: Data and task management ensuring reproducibility and tying together various parts of Salvus.

The experimental runs on EAGLE PRACE Tier-1 machine shows that Salvus job may require up to 64 GB of local memory and 1 GPU per node. It is possible to suspend to memory (relatively small) running jobs in order to free the nodes.

For the suspend / resume facility – SLURM scontrol management tool offers the appropriate commands [12]:

```
scontrol suspend job_list

    Suspend a running job. The job_list argument is a comma separated
list of job IDs. Use the resume command to resume its execution. User
processes must stop on receipt of SIGSTOP signal and resume upon receipt
of SIGCONT for this operation to be effective. Not all architectures
and configurations support job suspension. If a suspended job is
requeued, it will be placed in a held state. The time a job is suspended
will not count against a job's time limit. Only an operator,
administrator, SlurmUser, or root can suspend jobs.

scontrol resume job_list

    Resume a previously suspended job. The job_list argument is a comma
separated list of job IDs.
```

Before suspending, it might be good to check whether after freeing the node, the remaining memory will be large enough to run UC job.

The GPU suspend/resume facility is been provided by NVIDIA Virtual GPU (vGPU) - VMware vSphere Hypervisor (ESXi) [13]

During the internal discussion with other PRACE system operators, the GPU management is still on premature state and it is difficult to prepare a full virtualised stack of subsystems. That is the future, when the new machine will be deployed – the new paradigm of HPC resource management will be fully deployed (full HPC virtualisation).

2.3 KPI

Number of scientific codes was selected for testing in UC mode on at least one PRACE HPC system. The expected goal assumed the two different codes being tested on the selected machine. The work was conducted on EAGLE by use of HySEA and UCIS4EQ kernel which fulfills the requirement.

2.4 Future of this service

Nowadays, the natural hazards seem to become more frequent, spectacular and imply higher spending for removal of their effects. The future will definitely demonstrate the necessity of using computations in urgent scenarios on the selected systems.

2020/21 top Natural and Biological Disasters [14] [15] [16]

- The Australian Bushfire (burned an estimated 18.6 million hectares, destroyed over 5,900 buildings, and killed at least 34 people, while over 400 people were killed due to the residual smoke inhalation);

- Devastating Floods in Indonesia;
- Hurricane in United States (more than $60bn in damages);
- Volcano Eruption in The Philippines (The Volcano left huge ash clouds which caused mass evacuations of over 300,000 people);
- Earthquakes in Turkey, The Caribbean, China, Iran, Russia, Philippines & India (45 earthquakes characterised over 6 magnitudes. Jamaica and Russia were the worst hit with earthquakes over magnitude 7. The earthquake in Turkey claimed 41 lives);
- Locust Swarms in East Africa & Parts of India & Asia;
- Cyclone Amphan in India & Bangladesh;
- Europe Windstorm (Ciara and Alex that cost nearly $6bn and killed 30 people);
- Covid-19 pandemic;
- La Palma Cumbre Vieja volcano eruption (ash emissions, strong quakes, lava flowing, sulphur dioxide emission, thousands of people evacuated);
- Human migration due to climate change.

2020 was a historic year of extremes for the US according to NOAA's National Centers for Environmental Information (NCEI) report [17]. Hence, the attempt to reduce a disaster's influence on the people is inevitable and the UC price will not play the main role in the future.

The existing Urgent Computing use cases that are not granted access to the dedicated resources can benefit from using existing e-Infrastructures, making them valuable for the population. The present computational facility is being designed to have common uniform access, the same set of tools and similar bunch of scientific applications. Existing infrastructures have got the local policies that are in a contradiction with the requirements of urgent computing. The technological part of the new policy deployment is feasible according to the policy's guidelines being worked out in the legal tender between HPC center and various stakeholders.

The new paradigm of computational resource arises; except popular queuing systems on HPC machines, the cloud-based solution is gaining the existing areas of job runs. The authors of the application workflow must be aware of the characteristics of the environment where the urgent job will be placed.

PRACE has adopted this service and includes it into their portfolio. It is important to understand that SLA must agree in advance prior to the technical deployment of the scenario. There is no uniform scenario of urgent computing deployment. Each time, when a new request for urgent computing usage comes, a new separate tender must be made. The constraints are strictly in line with the particular application: e.g., FRTR (tsunami), UCIS4EQ (Seismic Workflow). This part of the tender belongs to the application developers and workflow architect. Summing up – the configuration of an entire Urgent workflow must be designed by scientific division of the application with the co-operation of future PRACE staff.

3 Service 2: in-situ visualisation

3.1 Description of the service

It is well known that more and more time is spent during a simulation for I/O operations and post-processing. I/O is recognised to be the main bottleneck to achieve the Exascale computing, which is not so far away. The first supercomputer in the TOP500 is the Fugaku system with 7,630,848 cores, which allowed it to achieve an HPL benchmark score of 442 Pflop/s [18]

Until recently, the in-situ analysis and visualisation field have been characterised by ad hoc, proof-of-concept prototypes that were, at the beginning, designed for monitoring and steering

the simulations, and only afterwards they were extended to execute in-situ tasks. The following tools fall in this category of ad hoc proof-of-concept prototypes: Cactus, CUMULVS, Damaris, EPIC, EPSN, Freeprocessing, Nessie, pV3, QIso, SCIRun, Strawman, VISTLE and yt. A brief summary of each infrastructure is described in [19]. However, there are some in-situ frameworks that caught our attention for the quality of its implementation and for the features they implement: Catalyst, Damaris and Melissa.

The aim of the project was to deploy these in-situ frameworks on at least three different PRACE systems and to evaluate their performance using some realistic use cases, with the perspective of making them a service available to the PRACE community.

Each infrastructure has been coupled with some CFD codes in order to be profiled. This has required to spend part of the project effort to develop the adaptors in the code language (e.g. Fortran, C++), or to improve the adaptor that has already been developed (e.g. OpenFOAM). The adaptor is a key software component to couple the CFD code with an in-situ infrastructure. It has to represent the simulation data into a format (e.g. VTK for catalyst) that can be managed by the in-situ framework and to read and execute in-situ pipelines (e.g. visualisation pipelines).

The work on in-situ visualisation done during PRACE-6IP can be summarised as following:

- We profiled the catalyst in-situ framework coupled with MIGALE, OpenFOAM and STREAmS on the GALILEO and M100 clusters. The catalyst framework is profiled both using a traditional cluster installation and using a singularity container;
- We profiled the Damaris in-situ framework coupled with Code Saturne on the Hawk cluster;
- We profiled the Melissa in-situ framework coupled with Parflow and WRF on the Juwels cluster.

In the next section we will describe the results achieved during PRACE-6IP. For each of the three in-situ frameworks, we provide a sub-section describing the use-case and profiling results, together with some information about the codes and the frameworks used in the service.

3.2 Results achieved during the project

3.2.1 In-Situ Visualisation using Catalyst

ParaView Catalyst [20] is an open-source data processing and visualisation library that enables in-situ, in transit and hybrid workflow. Built on top of and designed to interoperate with the standard visualisation toolkit VTK, Catalyst enables simulations to perform analysis, produce output data and visualise intermediate results during a running simulation concurrently [21].

In PRACE-6IP Paraview Catalyst has been tested and profiled with the following codes:

- MIGALE [22], a High Order Discontinuous Galerkin code developed by Francesco Bassi and Alessandro Colombo from University of Bergamo and Andrea Crivellini from Marche Polytechnic University;
- OpenFOAM [23], the well-known free and open source CFD software developed primarily by OpenCFD Ltd. since 2004. OpenFOAM uses the finite volume algorithm to solve the systems of partial differential equations over a complex domain that can be triangulated in an unstructured mesh of polyhedral cell. In this project, we still worked on the ESI version as a follow up of the work done in PRACE-5IP;

- STREAmS [24], an open source DNS CFD code of compressible turbulent flows in cartesian geometry, solving the unsteady, fully compressible Navier-Stokes equations for a perfect gas. Its main developer is Matteo Bernardini from University of La Sapienza.

The codes have been selected for the following reasons:

- A long and fruitful collaboration with the developers of the codes in EU projects;
- They implement different algorithms to discretize the Navier-Stokes equations and are used in different context: OpenFOAM is mainly used for industrial cases, due to its support for complex geometries and unstructured meshes, and the extensive range of features to solve complex fluid dynamics problems involving turbulence, heat transfer, chemical reactions and multi-phase. STREAMs is used for research CFD problems and handles only Cartesian structured meshes. MIGALE is a hybrid code, that targets both academic cases and industrial ones;
- All of them are CFD codes that have been tested and used massively on HPC clusters.

As mentioned earlier, the development of a Catalyst adaptor is crucial for the coupling of the CFD codes with the framework. Regarding MIGALE and STREAmS, WP6 co-developed the adaptor together with the main developers of the codes, in particular with Alessandro Colombo for MIGALE and Matteo Bernardini for STREAmS. Regarding OpenFOAM, WP6 used the updated version of the adaptor previously developed during the PRACE-5IP implementation.

The HPC clusters where we installed Paraview Catalyst are: GALILEO, MARCONI and MARCONI100. All these clusters are hosted at CINECA. GALILEO and MARCONI are CPU-based clusters with no GPUs in the compute nodes. For these machines, we installed a MESA enabled Paraview Catalyst version (version 5.6.3). On the contrary, M100 is an accelerated cluster which requires CUDA and EGL Paraview Catalyst version (version 5.8.1). We used GALILEO for profiling MIGALE and M100 for profiling OpenFOAM and STREAmS.

The profiling activities have been carried out to quantify the overhead of post-processing pipelines over the total time of the simulation using realistic use-cases. Below we go through the three CFD codes mentioned before.

3.2.1.1 MIGALE

In-situ visualisation has been tested on the T3L problem [[25], [26]], part of the ERCOFTAC test case suite [27]. For this test case the flow field is characterised at leading edge by a laminar separation bubble and, downstream the transition, an attached turbulent boundary layer. A mesh made of 38,320 elements with quadratic edges and a seventh-order discretisation P^6 was considered, resulting in 3.2M degrees of freedom per equation.

The in-situ approach allowed to clearly investigate the flow features development, e.g., hairpin vortices, at different values of the Reynolds number, i.e., $Re_D = \{1{,}725, 3{,}450, 6{,}900\}$, and free-stream turbulence intensity at the inlet, i.e., Tu = {0%, 2.3%}. Figure 1 shows some frames of the instantaneous iso-surface of the lambda2 criterion, coloured with the x-component of the velocity, generated via Catalyst for the different regimes. To assess the implementation of in-situ visualisation in code MIGALE, the Re_D=1725, Tu=0\% flow regime was considered.

To quantify the computational effort due to in-situ visualisation, the code MIGALE was profiled and the computation monitored over four-time steps for polynomial representations of the solution ranging from degree 4 to 6.

As an output of the visualisation process, a .png file is generated for each time step showing a plate with the surface mesh superimposed and identifying the vortex cores by means of an isosurface of the lambda2 criterion coloured with the x-component of the velocity, see Figure 1. For different polynomial degrees of the solutions, i.e., $P^{\{4-6\}}$, the cost of in-situ visualisation was monitored also varying the number of sub-elements $n_{se} = \{2,...,k+1\}$.

MIGALE has been profiled on GALILEO using 540 cores. The results are reported in Table 1. They show that the overhead of the in-situ visualisation strongly depends on the number of sub-elements used for the vtk representations of the solution, while it is less influenced by the polynomial degree. Although reducing n_{se} mitigates the computational cost of visualisation, lower values of n_{se} do not guarantee a resolution sufficient to appreciate some flow details, e.g. the downstream evolution of the hairpin vortices.

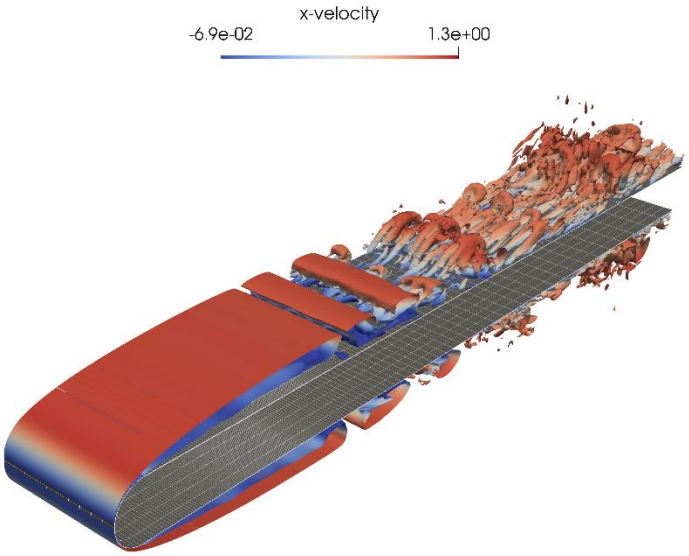

Figure 1: Instantaneous iso-surface of the lambda2 criterion coloured with the x-component of the velocity

K	N_{dof}	n_{se}	Tpipeline/Ttimestep
4	35	2	8.6%
		3	15.7%
		4	27.0%
5	56	2	13.4%
		3	28.7%
		4	53.4%
		5	90.2%
6	84	2	15.1%
		3	34.0%
		4	65.4%
		5	110.1%
		6	172.9%

Table 1 MIGALE - T3L: in-situ visualisation relative cost with respect to time spent for advancing the solution of a time step without in-situ [Tpipeline/Ttimestep] with the variation of the number of degrees of freedom [N_{dof}] and the number of sub-elements [n_{se}]

3.2.1.2 OpenFOAM

In this project, the delayed version of the X-LES (DX-LES) model has been used for the prediction of the pump impeller [28] performance of a centrifugal pump, both at the design- and one quarter-loads (off-design). The hybrid model can be seen as an improved version of the eXtra Large Eddy Simulation (X-LES) model [29]. The set of governing equations formally solved for X-LES consists in Reynolds-averaged Navier-Stokes (RANS) equations closed by a turbulent/non-turbulent (TNT) k-omega turbulence model, where the k-equation dynamically reduces from the k-equation of the turbulence model to the subgrid [30].

The simulations have been carried out with three different hybrid meshes (coarse, medium and fine) to investigate the correct spatial resolution needed to carefully describe the fluid features that characterise the impeller flow field at both operating conditions. In particular, the full geometry has been considered in order to remove the periodic assumption, that is not valid to study the instantaneous flow. Furthermore, a flow extension has been considered at the inlet section, to reduce possible disturbance induced by the blade leading edge. A diffuser has also been added to avoid recirculation at the outflow.

All computations have been performed using pimpleFoam, which is a transient solver for incompressible flows available in OpenFOAM, and the solution is advanced in time with the backward differentiation formula (BDF) scheme, a second-order implicit time-integration scheme. The simulations have been initialized with the RANS steady state solution (and shear stress transport (SST) k-omega turbulence model) with the Multiple Reference Frame (MRF).

The visualisation pipelines are generated automatically in Paraview and are coded in python, see Figure 2. They are built keeping in mind the quantities of interest to visualise in a centrifugal pump. The following two pipelines are used for the profiling:

- Q-criterion: this pipeline computes and renders the iso-surfaces of the Q-criterion quantity equal to 0.001. The iso-surfaces are coloured with the magnitude of the vorticity field together with the blade edges in the rendering process;
- 2D-streamlines: this pipeline computes the 2D-streamlines over a slice perpendicular to the inlet direction. The streamlines are coloured with the velocity magnitude together with the blade edges in the rendering process.

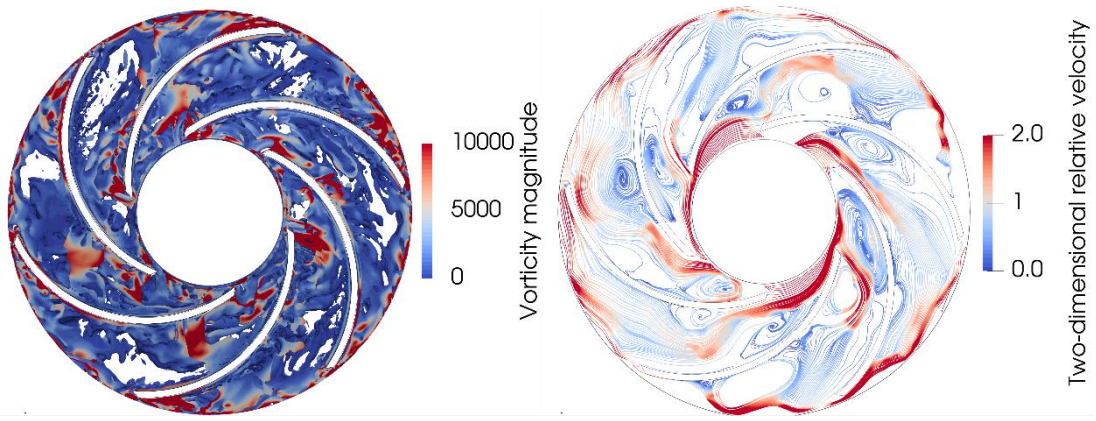

Figure 2: Q-criterion (left) and 2D-streamlines (right) visualisation pipelines

The results, shown Table 2 report the overhead of the execution of the two pipelines with respect to the elapsed time of ten iterations of the PIMPLE algorithm.

We performed six simulations per pipeline changing the size of the mesh (i.e. Coarse, Medium and Fine meshes) and the boundary conditions (i.e. Design and Off-design).

All simulations were performed on Marconi 100 using a number of nodes that keeps constant the computational load per core, 128 cores for coarse mesh, 192 cores for the medium mesh and 256 cores for the fine mesh. The profiling results reported in Table 2 have been achieved by running the simulations for 60 time steps, starting from a fully developed flow. In-situ visualisation tasks are executed every 10 time steps. The first 10 time steps are discarded while the other 15 are used for the average procedure.

According to the results of Table 2 we found that the PIMPLE elapsed time is faster in the design case with respect to the off-design and that it slightly increases from the Coarse to the Fine mesh. The Q-criterion pipeline is faster than the 2D-streamline but the in-situ elapsed time of each pipeline does not show a similar linear trend. However, the overhead of the in-situ visualisation pipelines over the PIMPLE elapsed time is low since it is in the range of 0.7-1.0% for the Q-criterion and 1-3% for the 2D streamline.

Case	PimpleTime	Pipeline	In-Situ time	Weight_10
Design coarse mesh	9.23s	Q-criterion	0.91s	0.99%
	9.26s	2D-streamline	1.11s	1.19%
Off-design coarse mesh	11.07s	Q-criterion	1.03s	0.93%
	11.10s	2D-streamline	2.75s	2.48%
Design medium mesh	11.02s	Q-criterion	0.87s	0.79%
	10.91s	2D-streamline	2.93s	2.68%
Off-design medium mesh	12.76s	Q-criterion	1.14s	0.89%
	12.51s	2D-streamline	3.19s	2.55%
Design fine mesh	11.45s	Q-criterion	1.01s	0.88%
	11.54s	2D-streamline	3.60s	3.12%
Off-design fine mesh	12.58s	Q-criterion	0.98s	0.78%
	12.78s	2D-streamline	3.17s	2.48%

Table 2: OpenFOAM – pump impeller: in-situ overhead of the Q-criterion and 2D-streamline pipelines respect to the elapsed time of ten iterations of the PIMPLE algorithm

3.2.1.3 STREAmS

The use-case selected in this project is the most complex of the three configurations that can be simulated using STREAmS, namely the shock-turbulent boundary layer interaction. The case analysed is the numerical equivalent of the experiment performed by [31], and the main parameters are set accordingly as: free-stream Mach number $M_\infty = 2.28$, Reynolds number of the incoming boundary layer (based on the momentum thickness) Re_{theta}=5,800 and $theta_{shock}$=8°, being $theta_{shock}$ the deflection angle of the flow crossing the shock.

The simulations have been carried out with three different Cartesian meshes (coarse, medium and fine), ranging from 0.13 to 8 billion. Time integration must be carried out for at least 2 million time-steps, to capture the significant low-frequency wall-pressure fluctuations associated with the boundary-layer shock-region interaction. The simulations are performed on the M100 cluster using a number of GPUs that keeps constant the computational load per GPU. The most important parameters of the simulations are reported in Table 3.

Case	Re	#points [billions]	#GPUs	dt [s]	Iteration time [s]
Coarse	235	0.13	8	0.0012	0.3
Medium	475	1.1	64	0.0006	0.3
Fine	950	8.6	512	0.0003	0.3

Table 3: STREAmS use-cases parameters: Reynolds number, number of GPUs, non-dimensionaltime-step, iteration elapsed time

The constant value of the elapsed time per iteration shows the ideal weak scalability achieved by the code, that runs completely on GPUs. From a visualisation point of view, we consider of interest six types of objects associated to variables and filters available in ParaView (see Figure 3 for an example):

- Density contours in an (x, y) plane (streamwise, wall-normal): visualise the evolution of the boundary layer including streamwise convection and development of turbulent vortices;
- Contours of streamwise velocity component in an (x, z) plane (streamwise, spanwise) at y=0.1 (wall-normal distance): this visualisation shows the turbulent structure in the core of the boundary layer;
- Contours of streamwise velocity component in an (x, z) plane (streamwise, spanwise) at the first y cell (wall-normal distance): since the velocity is zero at the wall, this visualisation shows the gradient field (proportional to skin-friction) along the wall-normal direction;
- Isosurface of Ducros sensor at value 0.3: identifies impinging and reflected shocks as visible surfaces in the three-dimensional domain;
- Contours of wall-pressure;
- Isosurface of swirling strength [32]. Useful for visualising turbulent eddies. The isosurface can be colored using values from another field, e.g., streamwise velocity component.

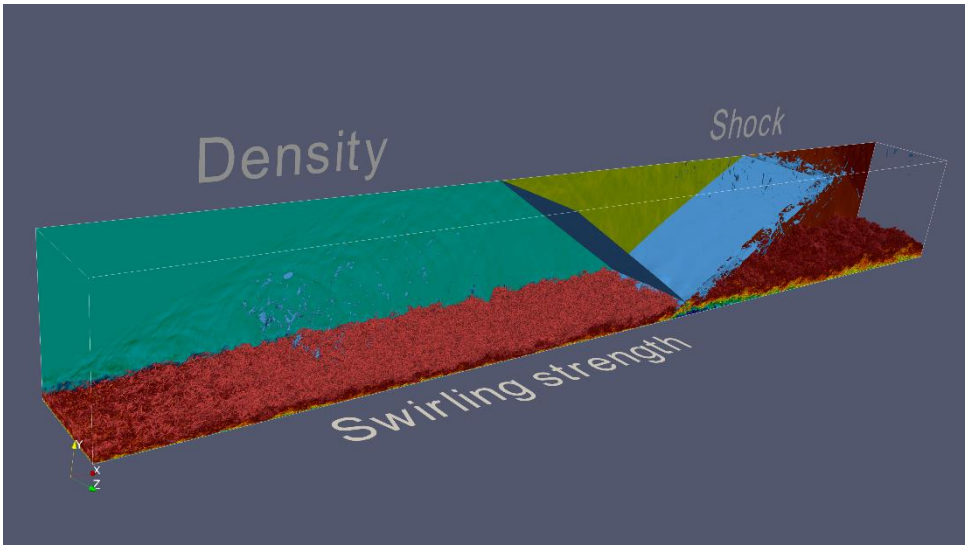

Figure 3: A visualiSation from full in-situ visualisation pipeline

The pipeline discussed here requires the use of two ParaView filters, namely Slice which extracts data along a flat surface of the domain, and Contour, which extracts the isosurface of a

certain quantity which can then be possibly coloured using the values of another quantity. For the Contour filter, which is computationally much more demanding, there is also an implementation available using the VTKm plugin capable, as seen, of exploiting the GPUs. The VTKmContour filter is potentially more performant and may in principle avoid data transfers from GPU to CPU.

The elapsed time of an in-situ pipeline generally includes the filter extraction time plus the additional time related to data preparation and image rendering. The results are shown in Table 4 for the coarse, medium, and fine cases. Elapsed times are averaged excluding the first in-situ iteration that includes the initialisation time.

Case	Coarse		Medium		Fine	
	Normal	**VTK-m**	**Normal**	**VTK-m**	**Normal**	**VTK-m**
Slice filter time	0.4s	-	0.4s	-	0.3s	-
Contour filter time	5.5s	3.3s	6.1s	3.7s	5.9s	3.5s
Additional times (data preparation, rendering)	1.4s	1.6s	1.9s	2.1s	3.3s	2.9s
Overhead Slice filter	7%	N.A.	7%	N.A.	10%	N.A.
Overhead Contour filter	23%	N.A.	27%	N.A.	30%	N.A.
Overhead Full Pipeline filter	37%	N.A.	40%	N.A.	49%	N.A.

Table 4: Elapsedd times [s] for in-situ visualisation pipeline sections: slice filter, contour filter, additional times (data preparation and image rendering). For contour filter VTKm is also considered

As expected, the Contour filter is much more computationally demanding than the Slice filter. Using VTKm substantially improves the contour performances, with percentage savings of around 40%. The scalability of filters in the spirit of weak scaling shows that there is a very modest variation of the times -- within 10% -- considering the different problem sizes. However, it is only valid for filters themselves while additional times -- i.e., rendering and data preparation times -- show a significant increasing trend when shifting to larger sizes (elapsed times more than double between coarse and fine cases).

To consistently evaluate the impact of an in-situ pipeline in the context of a STREAmS simulation, it is appropriate to consider a realistic scenario in which the image production occurs every n_{insitu} time iterations. Considering the explicit nature of the time integrator of the code, the time step is extremely small and, consequently, from the visualization point of view, it seems reasonable to consider values of n_{insitu} around 100.

The results reported in Table 4 show that the impact of the Slice pipeline is always very modest, i.e., around 10% in the worst case corresponding to the fine case. In contrast the impact of the contour pipeline is significant and reaches 30% for the fine case. For more complex pipelines, e.g. the full pipeline that includes the six visualisations described above, the percentage goes close to 50%.

3.2.2 In-Situ Visualisation using Catalyst in a Singularity container

This activity was based on developing a configuration script that would then serve as a recipe for building a container using the Singularity [33] development platform. The software installation within the container was done using the Spack [34] package management tool. As a base point, a CentOS image from the Docker repository was pulled. To have more control in the container building process, it was decided not to use containerize feature of the Spack as well as not to base the installation on the Docker image with the Spack pre-installed. The first step after pulling the image with CentOS 7.8.2003 from the Docker Hub was to install the prerequisites for the Spack using the yum package manager. The next step was to clone the Spack 0.15.3 from the repository and use it for all the further software installations. In the following, we report the list of the software stack installed using Spack:

- GCC 8.4.0
- OpenFOAM 1912
- Paraview 5.6.0 (Release version)
- Visualisation module for OpenFOAM provided by CINECA

Some of the additional tasks in the installation process were to determine the appropriate versions of the OpenFOAM and Paraview and the compilation flags for the further installation steps, and to configure the additional repository in Spack where the visualisation module provided by CINECA is located. After creating the container image, the performance tests were performed to see how the containerised application scales on a single node as well as on multiple nodes, with and without the visualisation process. For that purpose, the 'Wind Around Buildings' use case provided by OpenFOAM was used, with the number of cells refined (increased) to 1,481,887. The tests were done on the Galileo cluster at CINECA by using OpenMPI 3.1.6 and Singularity 3.6.1. Table 5 provides the results of the intra-node testing for 400 time steps, where visualisation occurs every 50-time steps. The overhead of the visualisation processing was in this case up to 38 %.

processes per node	number of nodes	wall time (with visualization) [s]	wall time (without visualization) [s]	Overhead [%]
2	1	212.95	194.09	9.72
4	1	121.14	118.14	2.54
8	1	103.89	80.98	28.29
16	1	79.14	62.46	26.71
24	1	67.88	53.65	26.52
32	1	64.22	48.92	31.28
36	1	63.42	46.13	37.48

Table 5: Results of the intra-node scalability testing

Table 6 provides the results of the inter-node tests where the use case was scaled across multiple nodes, with one process per node. The overhead is lower with respect to the previous benchmark, up to 16%.

processes per node	number of nodes	wall time (with visualization) [s]	wall time (without visualization) [s]	Overhead [%]
1	2	249.88	203.91	22.54
1	4	148.49	121.37	22.34
1	8	104.70	88.82	17.88
1	16	95.82	71.91	33.25
1	24	87.54	72.55	20.66
1	32	82.67	72.65	13.79
1	36	78.77	67.74	16.28

Table 6: Results of the inter-node scalability testing

3.2.3 Asynchronous In-Situ Visualisation using Damaris

The Damaris library [35] provides flexible, multi-use methods for MPI based HPC simulations to perform I/O in an asynchronous manner (I/O is referring to data output to disk or in-situ/in-transit visualisation process). Damaris allows a simulation to run with minimal interference with respect to the chosen I/O method. It allocates dedicated resources (either cores on a node, or dedicated nodes) which are used to carry out the processing. For in-situ visualisation, the Damaris library interfaces with both Visit LibSim and Paraview Catalyst (currently using the VTK library). This enables practitioners to have flexibility in the choice of the visualisation interface according to the availability software on a system and pre-configured scripting for output of the visualisation analysis. Until now, support has been available for particle, rectilinear and curved space grid models.

As a result of Task 6.2, the Damaris Paraview interface provides support for the unstructured-grid mesh type. The development of the unstructured mesh model adaptor was addressed by the use-case of integrating Damaris with Code_Saturne. Code_Saturne [36] is an open-source, flexible, highly scalable CFD code that is well known within PRACE and is one of the standard benchmarking targets of the PRACE Unified European Applications Benchmark Suite (UEABS). The combined Code_Saturne + Damaris implementation has undergone testing on the HLRS Hawk system in Stuttgart, Germany. The results of the tests have been described into the PRACE white paper WP310 [37]. A brief overview of the results will be presented here.

The Hawk supercomputer is an AMD Rome CPU based system, using 2 64-core sockets in an 8-way NUMA configuration within a node. The nodes are connected in a 9D hypercube topology via a 200 Mb/s HDR Infiniband network.

The models developed for testing Code_Saturne were designed to be of the size expected of high-resolution industrial models and ranged from 29M to 115M hexahedral elements. The simulations were run from 1 node to a maximum of 32 nodes (i.e., 128 – 4,096 cores). Simulation rank placement testing indicated that hybrid MPI+OpenMP configurations are optimal, and three OpenMP threads per rank and 32 ranks per node was chosen as an optimal distribution.

We developed three in-situ visualisation pipelines using ParaView Catalyst. They performed relatively simple processing, taking cross sections of the model (one or two cross-section, three fields, that scaled with model size) and saving to CSV based format, or taking a single cross-section and a full 3D field (velocity) and saving to VTK XML based format.

Due to the large number of cores per node on the Hawk system, some optimisation of placement of the Damaris cores was found to be important. The white paper (WP310) discusses two

choices of placement made in detail. This work also identified that placement options for cores should be added to the Damaris configuration file since it would enhance usability and make using Damaris less dependent on the system based, non-standardised methods of process placement. As part of the core placement work, the effect of changing the number of cores allocated to Damaris was also explored and is another area where added usability (Damaris server core "free time" logging) would help the users in finding optimal configurations for numbers of cores. The two flagged additions to Damaris should be implemented in the next Damaris release (v 1.6) and support for unstructured mesh models with Visit LibSim is also planned for an upcoming release.

The results from the Hawk testing of the Damaris integration with Code_Saturne have been presented at the Code_Saturne User Meeting 2021, at EDF in Saclay. Discussion with the developers of Code_Saturne at the EDF R&D, Yvan Fournier and Chai Koren, has commenced about having Damaris as a supported library in a future Code_Saturne release.

The white paper also discussed the amount of code changes required to add Damaris support to Code_Saturne. One large addition was the transfer of the unstructured mesh data to Damaris. This should be a highly reusable procedure, that could be incorporated into the Damaris library itself. Another was the development of the "fvm" (finite-volume) template I/O structure that Code_Satune uses to have a modular I/O library sub-system. This is where the simulation field data is passed to Damaris on each requested iteration. Overall, the code additions were limited. A higher proportion of code additions were made to the Damaris back-end to support the unstructured mesh data, although again this addition only required an extra member inheriting from the C++ `Damaris::Mesh` class.

Some representative scaling results for the 58M cell model are presented in Table 4. The graphs show strong scaling results for the three Paraview Catalyst pipelines used in the study. The red boxes are distributions of timing for the Code_Satune runs, without using Damaris. The Blue boxes are timing distribution quantiles for in-situ processing with Damaris (i.e. processing is asynchronous to the simulation) and the green boxes are distributions of timings for iterations that did not request in-situ processing (even iterations). In each pipeline the simulation timesteps using Damaris are equivalent (between pipelines) and in all cases less than the times when in-situ visualisation was done synchronously. The Damaris times are seen to have a small overhead compared to no in-situ output iteration data. Also presented on the graphs are the times taken for processing on the Damaris server (circle with cross). These are average times as individual iteration times for the server are not available in the log data. It can be seen that for the 2xCSV slice output, using 4 nodes onwards, the server times end up greater than the simulation iteration times. This indicates that the server nodes are overloaded and either more resources should be given to them for processing, or less work should be attempted per iteration. This kind of overloaded system can result in data loss as Damaris will drop buffered data without processing if it runs out of buffer space, so generally should be avoided.

The white paper goes on to show that in-transit processing can improve performance of the high load pipeline. A practical take away of this is that use of optimised I/O routines (such as the VTK XML output) is necessary to get the best scaling performance of a system, whether it is processed asynchronously or not.

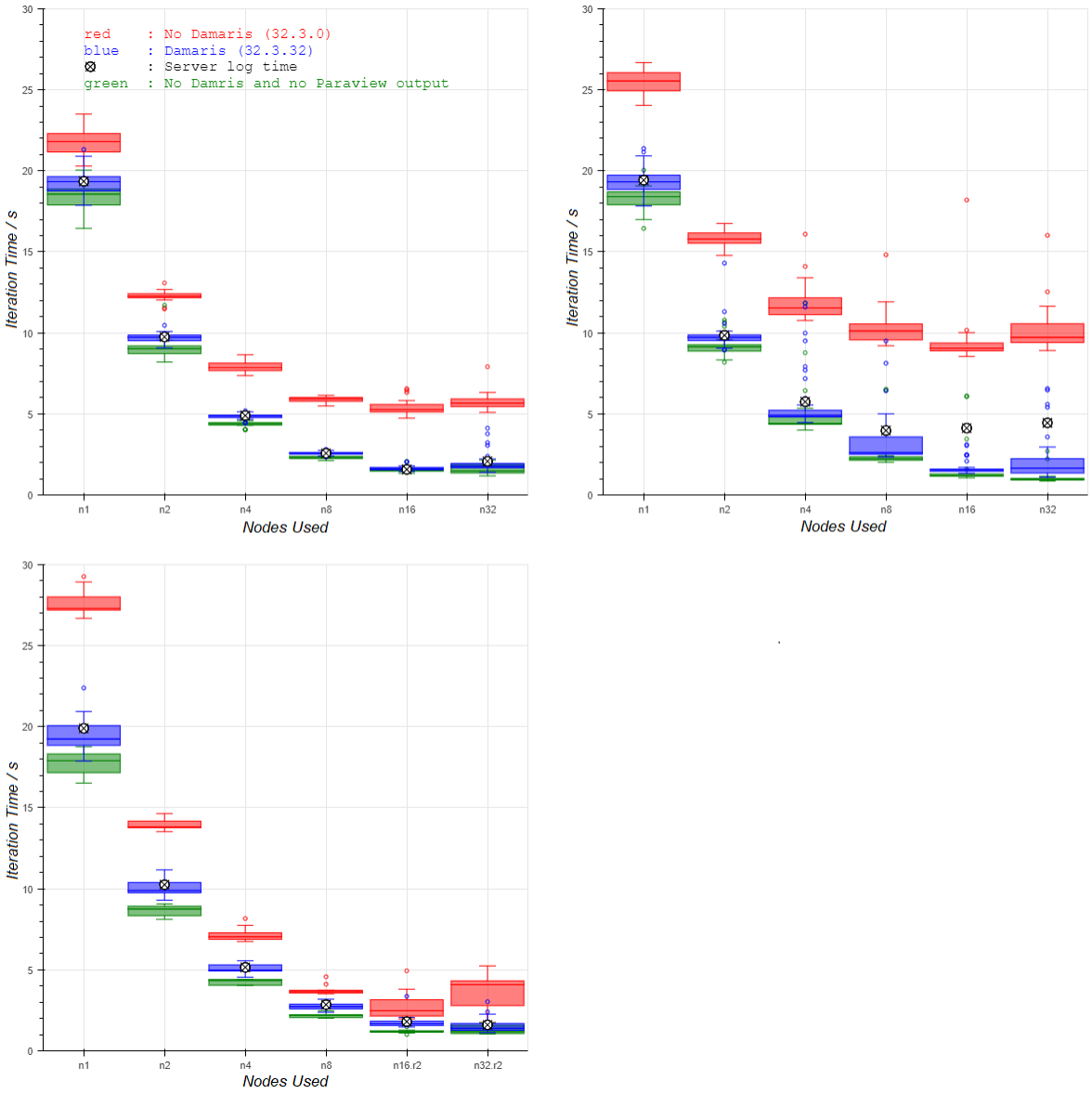

Figure 4: Box and whisker plots of strong scaling for 58M model, single CSV output (top left), double CSV output (top right), XML output(bottom left). All runs are using 32 processes per node and 3 threads per process. Small coloured points are outliers.

To conclude, the Damaris implementation of asynchronous in-situ visualisation for Code_Saturne was completed successfully. The implementation and testing carried out during the project enhanced the capability of Damaris by adding unstructured mesh model support and highlighted usability improvements. The profiling results on the Hawk machine show that the Damaris library is a good candidate for adding in-situ visualisation support and improving code scalability by performing the in-situ analysis asynchronously. The asynchronous processing must be carried out with some planning according to the resources to be used, to fit the finite time available for doing the required processing. Tuning the number of computational resources and their placement within a node is required, and selection of appropriate algorithms for analysis and efficient I/O routines for saving data is important so as not to overload the available Damaris resources. Damaris allows for in-situ and in-transit processing providing further options for a simulation to obtain the required processing capacity. The resulting white paper

documented the work and how to effectively use Damaris, thus supporting users and engineers wanting to implement and use Damaris based processing.

3.2.4 In-Situ Visualisation using Melissa

Most in-situ processing solutions focus on enabling on-line data processing capabilities for a single simulation run instance. Ensemble runs that consist of running several instances of the same simulation code, with different parameters, are getting more common as the computing power available is increasing. This kind of approach is also referred to as parameter sweep Monte Carlo methods. Methods relying on ensemble runs include sensibility analysis, training of deep surrogate models, data assimilation, and reinforcement learning to name a few. The amount of data generated by an ensemble run, that can consist of thousands of simulations runs, each one being already a complex advance parallel simulation code, is quickly overwhelming, saturating the system storage, slowing down the simulation executions, but also other users' runs impacted by interference on the I/O system. In–situ solutions, or more precisely in transit data processing ones, are thus even more critical in that case than for a single simulation run.

Melissa [39] is an open source (BSD license) framework dedicated to large scale ensemble runs and on-line/in transit data processing. Melissa has been developed with built-in features that are essential when targeting Exascale: fault tolerance and elasticity. As an example of Melissa capabilities, largest runs for sensibility analysis study so far involved up to 30,000 core, executed 80,000 parallel simulations, and generated 288 TB of intermediate data that did not need to be stored on the file system.

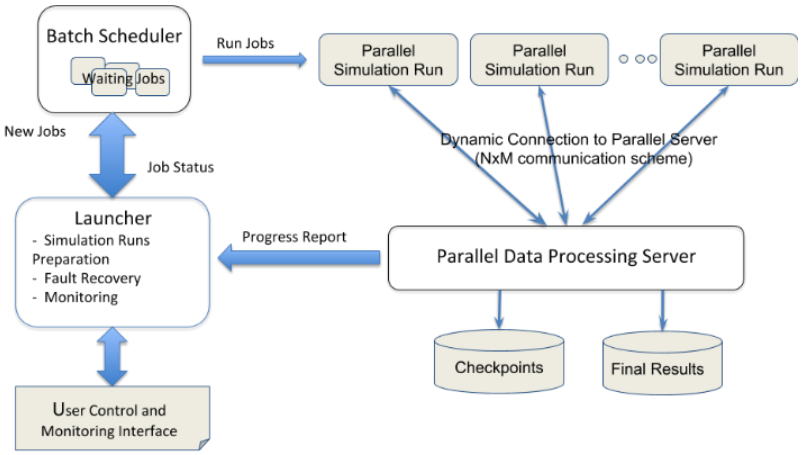

Figure 5: Melissa Architecture Overview

The Melissa architecture relies on three interacting components, see Figure 5:

- Melissa runners (clients): the parallel numerical simulation code turned into a client. Each client sends its output to the server as soon as available. Clients are independent jobs;
- Melissa server: a parallel executable in charge of iterative (on-line) data processing. Upon reception of new data from any one of the connected clients, the server processes it to update its on-line computations and discard it, so the memory requirements on the server side stay under control;
- Melissa Launcher: the front-end Python script in charge of orchestrating the execution of the ensemble run. This is the user entry point to configure the study of experiment.

The launcher is also tightly interacting with the machine batch scheduler to control executions, including faults and elasticity.

Today Melissa supports two types of applications directly:

- **Sensibility analysis** where the parallel server computes statistics relying on parallel incremental algorithms (for computing the average, variance and co-variance, skewness, kurtosis, minimum, maximum, threshold exceedance, quantiles and Sobol' indices);
- **Data assimilation** where the server implements a parallel EnkF filter or a particle filter, with capabilities for load balancing member propagations to clients.

Melissa related efforts have focused on:

- Consolidating testing and continuous integration:
  - Extension of the test suite and continuous integration running with several linux distributions, supporting several processor architectures (i386, amd64, arm64), including CentOS which is very similar to the RedHat Enterprise Linux distributions found on most supercomputers;
  - Development of a virtual cluster based on LXD containers. This enables to set-up a virtual multi-node cluster with a batch scheduler (OAR and Slurm supported) to test Melissa in a more realistic context. Each node (compute nodes and front-end node) is a LXD container. A Melissa execution can be started from the front-end node exactly as on a supercomputer, using OAR or Slurm for resource allocation. This enables to setup distributed functional tests. This is an important tool as Melissa interacts tightly with the batch scheduler. In particular, the virtual cluster enables to test various failure scenarios that involve scheduler dependent code [38]. The virtual cluster is independent from Melissa and can be reused by other projects;
- Spack package. Spack [34] is emerging into a standard package manager for supercomputers. Spack enables an easy installation by users, considering available modules. Spack does not even require specific installation on the machine, as Spack get installed through a simple “git clone” and Spack package installation does not require root access as sometimes required by other package managers (Nix for instance). Within this service, we developed Spack support for Melissa. It has been integrated in the Spack distribution by Spack developers [39] We also developed a Spack package for the Code_Saturne CFD code, the base of an advanced example for Melissa, that has been submitted to the Code-Saturne development team. The combination of both enables an easy install and deployment of large-scale sensibility analysis with Code_Saturne;
- Testing of Melissa on different supercomputers:
  - Jean-Zay (France);
  - Jewels (Germany);
  - Marenostrum (Spain).
- Testing at scale: Leveraging the developments done in the context of the project Energy Oriented Center of Excellence: toward exascale for energy (EoCoE-II), leading to an operational version of Melissa for data assimilation with two flavours, we led scaling tests on Juwels and Jean-Zay supercomputers:
  - Data assimilation with EnKF, with the ParFlow simulation code, relying on Parallel Data Assimilation Framework (PDAF) as the data assimilation engine, propagating 16,384 members on 16,240 cores;

- Data assimilation with Particle Filter, with the WRF (Weather Research and Forecasting Model) simulation code, an in-house particle filter implementation, reaching 2,555 members on 20,442 cores.

3.2.5 KPI

We identified at the beginning of the project the following KPI as an interesting indicator of the project progress: *Number of PRACE HPC systems that enable in-situ visualisation coupled with some scientific codes.*

This KPI was monitored since October 2020 with numbers checked every month. We set up a minimal requirement of 3 HPC systems to be in-situ enabled.

This requirement has been satisfied by first installing the in-situ frameworks involved in the project (Catalyst, Damaris, Melissa) into the following HPC PRACE clusters:

- Catalyst installed at:
  - CINECA (GALILEO, MARCONI, MARCONI100)
- Damaris installed at:
  - HLRS (Hawk)
- Melissa installed at:
  - Jülich Supercomputing Centre (JEWELS)
  - BSC (MareNostrum)
  - IDRIS (Jean-Zay)

The next step was to select some scientific CFD codes of relevant importance in the CFD community to be in-situ instrumented. We developed a coupling interface or improved an existing one for the following codes:

- MIGALE
- OpenFOAM
- STREAmS
- Code Saturne
- ParFlow
- WRF

The first three codes have been coupled with Catalyst, the fourth with Damaris and the last two with Melissa. All the codes have been tested using a real HPC use-case and the first four have been profiled in order to evaluate the in-situ visualisation relative cost with respect to time spent for advancing the solution of a time step without in-situ.

3.3 Future of this service

The usefulness of an in-situ service in the context of high-fidelity CFD simulations is largely recognised. Such service will be more and more useful in the upcoming exascale machines. Moreover, not only the CFD context can get benefits from such a service.

The future of this service is surely related to software maintenance activities. The software stacks of the in-situ frameworks tested in PRACE-6IP (Catalyst, Damaris, Melissa) needs to be updated in the HPC sites where they have been installed. Since the clusters are substituted with a frequency of 3-5 years in a HPC site, the software stack should be installed in all the upcoming

machines. Last but not the least, the adaptors of the instrumented codes (MIGALE, OpenFOAM, STREAmS, Code Saturne) of PRACE-6IP needs to be maintained as well in order to be coupled with the next versions of the in-situ frameworks and to run in the upcoming machines.

This service can also have a future for development activities and for testing new technologies. New frameworks can be tested and compared with the ones proposed in this project. New CFD codes largely used in the scientific community can be instrumented in order to expand the area of influence of such technology.

Regarding a Melissa related service, a further consolidation for in transit data processing for large scale ensemble run requires effort in:

- Complying with the policy of the Extreme-scale Scientific Software Stack initiative [40]. Spack was the first step. If Melissa manages to be integrated into E4S, this will ensure Melissa be validated on a wide variety of supercomputers;
- Extend support to ARM (A64FX in particular) and Risc-V architectures, as part of next generation European suppercomputers will likely be based on such architectures. Notice that we recently got access to the newly installed TGCC A64FX partition (France), and that early tests with Melissa are currently in progress on Fugaku (Japan).
- Extend the data processing capabilities, in particular with a Python based server supporting TensorFlow and Horovod for parallel training, for on-line training from ensemble generated data.

Today the EuroHPC JU has procured seven supercomputers, located across Europe. Two of them are top-of-the-range supercomputers, Leonardo capable of executing more than 248 Petaflops and LUMI more than 375 Petaflops. These HPC machines are paving the way towards the exascale frontier, i.e. supercomputers that can perform more than one trillion operations per second.

The cluster Leonardo will be hosted at CINECA and it represents a perfect candidate for testing the codes we have instrumented in PRACE-6IP. In this new leadership-class HPC system, the choice of performing in-situ tasks during the simulation, e.g. in-situ visualisation, will be of primary importance. A similar service we have developed in PRACE can be hosted at Leonardo and used for simulations that are able to exploit the huge computation power provided by its hybrid architecture.

4 Service 3: The deployment of containers and fully virtualised tools into HPC infrastructures

4.1 Description of the service

Linux containerisation is an operating system level virtualisation technology that offers lightweight virtualisation. An application that runs as a container has its own root filesystem, but shares the kernel with the host operating system. This has many advantages over virtual machines. First, containers are much less resource consuming since there are no guest OS. Second, a container process is visible on the host operating system, which gives the opportunity to system administrators for monitoring and controlling the behaviour of container processes. Linux containers are monitored and managed by a container engine which is responsible for initiating, managing, and allocating containers. Docker [41] is the most popular platform among

users and IT centres for Linux containerisation. A software tool can be packaged as a Docker image and pushed to the Docker public repository, Docker hub, for sharing. A Docker image can run as a container on any system that has a valid Linux kernel. HPC targeted platforms, e.g. Singularity [42] and Sarus [42] , make it possible to use Docker containers in production for HPC systems.

Unikernels are lightweight single application operating systems developed for the cloud, edge computing, Internet of Things, etc. They fit into small images, have low memory foot-print, and boot in less than one second. They are known for accelerating the execution of programs and improving throughput of network applications. This technology has proven its qualities for theses domains with numerous unikernels and publications [42] , [43], [44] , [45] , [58] .

This service is an extension of the lightweight virtualisation service in PRACE-5IP where it has targeted evaluation and benchmarking of containerised and fully virtualised workloads on both bare-metal and cloud-based HPC clusters in terms. The following conclusions have been drawn:

- **Security:** Security issues have been investigated and are resolved in most container platforms;
- **Scalability:** Scalability tests, performed using singularity with MPI, proved that container applications are scalable. Unprivileged container platforms, uDocker and Singularity, proved to be scalable in terms of deployment;
- **Performance:** There are almost no performance issues (especially for containers).

In the PRACE-6IP project, this prototypal service has been a collaboration of several PRACE sites: UiO/Sigma2, CINECA, CESGA, CSCS, UHeM, UL, CEA, LRZ, and IT4I. After evaluating the eligibility of containerisation technology for HPC in general and suitable container runtimes in PRACE-5IP, in PRACE-6IP the focus has been to evaluate more recent container runtimes “in operation” for HPC applications, in addition to investigating the usefulness of unikernel technology for OpenMP applications. Additional container runtimes to the previously evaluated Docker and Singularity are: Sarus and Charliecloud [44] [43]. The production support of Singularity has also been evaluated. Unikernels have been evaluated for HPC workloads (mainly OpenMP). Containerised HPC using elastic composition of HTCondor clusters using Docker swarm has also been tested and evaluated.

It can be fairly concluded that Singularity, Sarus, Charlicloud runtimes are mature and can be used in large scale HPC applications with minimal performance overhead. However, containers abstract all layers “above” the kernel, not below. This means that, for containers to run efficiently on modern processors and GPUs etc. the proper drivers need to be installed and correctly configured. However, due to using the host kernel, containers use a lot of system calls. This is not the case for unikernels. Unikernel is full virtualisation which provides VM-like isolation and runs on the top of a hypervisor. They convert system calls to function calls, which makes them more secure than containers. In addition, Unikernels are lightweight. But they are still lacking maturity. Currently there are no GPU/IB drivers supported for Unikernels. Promoting unikernels for production use is still early.

4.1.1 Sarus

Sarus [42] is a container engine for high-performance computing (HPC) systems that provides a user-friendly way to instantiate feature-rich containers from Docker images. It has been designed to address the unique requirements of HPC installations, such as: native performance from dedicated hardware, improved security due to the multi-tenant nature of the systems,

support for network parallel filesystems and diskless computing nodes, compatibility with a workload manager or job scheduler.

Sarus relies on industry standards and open-source software to implement a modular architecture and to extend the capabilities of the container runtime through external plugin programs, called "OCI hooks". Hooks can customize containers to enable specific high-performance features, and they can be developed independently by third parties to introduce new technologies or improve support for existing ones.

4.1.2 Charliecloud

Charliecloud provides user defined software stacks for HPC systems. It uses Linux user namespaces to run containers with no privileged operations or daemons and minimal configuration changes on center resources. The Charliecloud container images can be built from Docker images with a set of simple commands in a terminal window.

Documented deployment methods

For deployment on HPC systems, the initial step is to create a Charliecloud image from a docker image on a local workstation (this requires root rights not available to regular user accounts on LRZ systems), and then copy that image to the target HPC system.

An LRZ-provided installation of Charliecloud that can be used to start up the container on the LRZ HPC systems. For production purposes, the startup procedure is usually submitted as a SLURM batch script.

Using SLURM and Intel MPI, parallel execution of applications installed in the image is supported. Furthermore, it is possible to

- Make host file systems visible and usable inside the image;
- Set up the environment in the container in the same or similar manner as for the host (including the use of the host's environment module system), if so desired.

4.1.3 Unikernels

Unikernels are usually compared to containers because both approaches are a form of lightweight virtualisation. However, unikernels have more performance benefits than containers. They also reduce the attack surface by including only the kernel code required by the executed application. Because unikernels increase the throughput of some applications, they may improve HPC applications' performances. Reducing OS-noise could also improve the synchronization between threads inside a single node parallel application, and between nodes in MPI applications, leading to an acceleration of their execution. The goal of this study is to evaluate what are the benefits that current unikernel technologies can provide in an HPC context.

Unikernel Specialisation

Because there is only one application running, unikernels execute all code (including user code) in supervisor mode (ring 0 on the x86-64 ISA). This allows to speedup system calls performed by the application. System calls are known to be a slow operation because they include user/kernel world-switches [47]. Indeed, before performing the system call, in a traditional OS a user program is being executed in user-mode. When the system call instruction is executed, the user-mode execution is stopped, and the processor switches to kernel (supervisor) mode. Once the kernel has completed the processing of the system call, it returns its results and

switches back to user-mode. In common processors, this mode-switching requires flushing the CPU pipeline, saving standardised registers (depending on the system call performed) onto the stack, changing protection domain, etc. This process of switching back and forth from user-mode and kernel requires many CPU cycles [48] . Also, since the discovery of the Meltdown vulnerability in processors older than 2018, a new feature has been added in regular kernels that is slowing down system calls even more. The KPTI (Kernel page-table isolation) feature involves a page table switch each time there is a mode-switch, implying a costly TLB (translation lookaside buffer) flush. With unikernels, the application runs in kernel-mode and the mode-switching of system calls is not required. This means that concretely the system calls are common function calls, saving many CPU cycles. Unikernels have their own way of transforming system calls. For example, OSv and HermitCore links the application against a custom libC that calls directly the kernel functions instead of performing system calls [49], while HermiTux uses binary rewriting to update the system call invocation instruction into function call for statically compiled binaries [50]. Running all code in kernel mode results in a complete lack of memory isolation within a unikernel instance. However, isolation with the other applications, i.e., with other unikernels/virtual machines, is provided by the hypervisor.

Unikernel Compatibility

Compatibility with existing applications is an important matter with Unikernels. Some unikernels (e.g., HermitCore [49] Rumprun [51]) will require the source code of an application to be recompiled in order to execute it. They are said to be source-compatible and aim to require low to no modifications of the application's code. By recompiling the application, the unikernel's features are linked directly to the target application. The generated binary executable contains both the application and the unikernel OS. In this case, the compatibility between application and unikernels is handled at compile-time. If a feature is not supported by a unikernel, the compilation of the application will generally fail. In this case, the developer will need to port an application for the targeted unikernel so that it will compile with the unikernel. Another way of handling compatibility between applications and unikernels is by providing binary compatibility. Some unikernels (e.g. HermiTux [51] , OSv [52], Unikraft [53]) are able to execute Linux executable binaries compiled for a popular OS, Linux. In this case the compatibility is handled by the unikernel, which needs to support the features called by the binary executable. If a feature is not supported, the application will crash or misbehave at runtime. This time it is the responsibility of the unikernel's developer to implement the missing feature in order to get the executable running. If the unikernel maintainer can not implement the feature for some reasons, there is still a possibility to try to avoid using the feature.

Unikernel Models

Unikernels can be classified in two categories: Language-based unikernels, and POSIX-Like unikernels. Language based unikernels are tied to one specific programming language. Their degree of compatibility is low as they generally only support applications written in this language, and require the application to be written specifically for their own API (Application Programming Interface). The concept of "OS as a library" is very well adapted for the seskernels, because they really are a set of libraries that need to be included in the application source code. Although they provide a very good optimisation and specialisation of the kernel, the major downside of these unikernels is that they require more development effort to port an application. Language based unikernels are for example MirageOS [54] an OCamL based unikernel, and IncludeOS [55] a C++ based unikernel. Contrary to language-based unikernels and their specific API, POSIX-like unikernels try to provide the most complete implementation of the POSIX API they can. These unikernels offer some compatibility at the source level for Linux applications. However, because their implementation of the POSIX API is often incomplete, they sometimes require porting efforts to execute a given application. In the best cases, only the recompilation of the application sources is required, while in the general case a

few porting efforts are required to make it run. Examples of POSIX-like unikernels are: HermitCore [48] HermiTux [50] Rumprun [51] OSv [52], Lupine Linux [46] and Unikraft [52].

Unikernel Analysis in the Context of HPC

This section presents our analysis of several unikernels with respect to their support for HPC applications. By HPC applications, we restrict this study to applications meeting the following criteria:

- They are compatible with C, C++ and FORTRAN applications;
- They can run on multi-core architecture;
- They are compatible with OpenMP applications.

The support of languages, multi-core, and OpenMP can be verified by compiling and executing a basic OpenMP application that spawns several threads that iterate over a loop. If the compilation succeeds, the process monitorhtop can be used to verify that the threads are really running simultaneously on distinct cores of the machine. If the application can be compiled and can complete its execution, we consider that the unikernel meets our HPC criteria.

Specific Hardware Support

Another criterion that was considered is the support of existing drivers for specific HPC hardware. Some HPC applications are able to take advantage of dedicated architectures, such as high bandwidth network cards or GPGPUs. The support of these particular devices is achieved by specific drivers that are often proprietary. However, the field of driver support has not been explored by many unikernels. Since they were originally designed for cloud and embedded contexts, the mechanisms for driver support have not been implemented. In the unikernels we considered for this report, only Rumprun and Lupine Linux may be compatible with Infiniband and GPGPU drivers. For other unikernels, the drivers would have to be reimplemented from scratch directly in the kernel. This is something we could not afford for this study. Hence, we decided to leave this criteria for future work.

Unikernel Analysis

The following study is restricted to POSIX-like unikernels, to avoid rewriting applications for distinct APIs. By proceeding this way, we greatly reduced the amount of work needed to run the benchmarks with unikernels: we only had to make a few modifications to port them for the selected unikernels. The following details six unikernels: OSv, Rumprun, Unikraft, HermitCore, HermiTux and Lupine Linux unikernels.

OSv [53] is a unikernel that was originally developed by Cloudius Systems, interfacing with the application at the C Library level. It uses a custom C library based on musl C library2. This custom libC is designed to avoid making system calls by directly calling OSv's kernel functions. It aims at efficiency by implementing lock-free algorithms, per-CPUs waiting queues for threads, tickless thread scheduling and a lightweight network stack. OSv provides compatibility with the Java Virtual Machine, which makes it compatible with Java applications. The project is now open-source, and some volunteers still commit changes to the OSv repository. OSv support C, C++ and FORTRAN applications however, it does not support OpenMP applications. So, we chose to put it aside.

Rumprun [51] is an unikernel developed by the FreeBSD foundation [57], but not maintained anymore. It is based on the NetBSD Rump anykernel. It is a set of drivers and system call handlers. Rumprun can run on any platform able to execute C99 code, with only a hundred kilobytes of RAM/ROM. This unikernel can run on bare metal, or above a hypervisor. Because Rump kernels were originally designed for developing drivers, we believe that this kernel would be a good candidate for experimenting with HPC drivers. However, despite its low-level architecture compatibility, Rumprun is not compatible with SMP execution. To do so, it

requires a ”multi-kernel” approach: Spawning a unikernel on each core, and making them communicate through IP-protocol. Also, Rumprun only supports C/C++ applications. These points do not meet our criteria; therefore we eliminate Rumprun from our list.

Unikraft [58] is a unikernel developed by NEC laboratories. It is designed to be fully modular and cus-tomizable, and very efficient due to performance-minded and well designed APIs. It tries to improve application performance in two ways. The first method is the gain offered by the unikernel paradigm. It reduces the overhead of systems calls, the memory footprint of the kernel and can accelerate memory allocation by choosing the right allocator for an application. The second way of performance improvement can be achieve by adapting the application to take advantage of Unikraft's specialised APIs, where performance is critical. Unikraft is supported by a strong developer's community, therefore it is a great candidate for our study. However, it does not support multi-core yet. Therefore, it is not considered in our study, but it will surely be worth looking at in a few years.

HermitCore [49] is an unikernel developed at RWTH Aachen University (Germany), and designed for extreme-scale computing. It is designed with performance in mind, and aims at reducing overhead caused by the OS. It uses the New lib3C library, a library originally designed for embedded systems, requiring only a few system calls from the OS. It uses a dynamic timer to avoid most of the interruptions of the running application - and its threads. It supports multi-core applications and OpenMP C/C++ and FORTRAN applications, so we use it in our study. It is important to note that even if HermitCore is compatible with OpenMP application (thanks to an Intel OpenMP runtime shipped with the unikernel), it is not compatible with every OpenMP runtime. In this study, we consider the "original" HermitCore unikernel, written in C language. It is not actively maintained anymore as the development effort has shifted to the Rust version of the unikernel.

HermiTux [51] is an unikernel based on HermitCore, developed at Virginia Tech (USA) and the University of Manchester (UK). It has been designed to be a binary compatible with Linux executables. It does not need recompilation of Linux applications before executing them. By doing so, it reduces the effort required to port applications. HermiTux reduces the system calls overhead thanks to an optimised system call handler. Even if the unikernel is not maintained as often as it was before, it keeps getting support from its main developer. HermiTux supports multi-core as well as OpenMP applications. Because of its binary compatibility with Linux software, it is compatible with C/C++ and FORTRAN applications. This binary compatibility is more permissive for controlling compilation and linking of OpenMP runtime. Hence, we use it in our study of unikernels.

Lupine Linux [48] is a configuration of the Linux kernel developed by the University of Illinois (USA) and IBM research. It specialises the Linux kernel for the execution of a single application, and removes what is unneeded in the original kernel for a given application. Lupine Linux is not actively maintained anymore. Because it relies on the Linux kernel, we could expect a very good compatibility for applications and drivers. Unfortunately, because the configuration of the kernel can be time-consuming, we did not had enough time to complete our analysis of Lupine Linux and study if we were able to run multi-core OpenMP applications with it.

Discussion

Table 7 sums up the evaluation of unikernels with respect to our criteria. The main conclusion is that the support of multi-core and OpenMP is not a very common feature in unikernels. Unikernels have been designed to be suited for the cloud ecosystem rather than the HPC context. Because of their lightweight and quick boot time, unikernels tend to be spawned multiple times on different cores rather than have a unique instance using several cores. A

second conclusion is that HermitCore and HermiTux met our three criteria and are not too complex to be handled. The next section describes how to install and use them.

unikernel	C/C++/FORTRAN support	multi-core support	OpenMP support
OSv	**yes**	**yes**	no
Rumprun	no	no	**yes**
Unikraft	**yes**	no	no
HermitCore	**yes**	**yes**	**yes**
HermiTux	**yes**	**yes**	**yes**
Lupine Linux	**yes**	-*	-*

*: Not evaluated.

Table 7: HermiTux and HermitCore are the only unikernels that meet our criteria

4.2 Results achieved during the project

This section presents the results and experience for deploying and operating several container runtimes on different PRACE sites, in addition to benchmarking of Sarus runtime and several unikernel platforms for OpenMP.

4.2.1 Charliecloud

The following experiences have been collected:

Stability

To ensure stability especially at scale it is important to remember that the container is running on the HPC system and that the containerised workflow is not performing some type of operation that would normally cause the HPC problems. In addition, it is important to use the underlying HPC systems MPI environment.

Resource overheads

The typical overheads of running the HPC container is negligible in terms of memory and performance compared to running the workflow directly on the HPC system. However, issues can arise if the workflows memory requirements are pushing up against the amount of memory available on the node. Or they are submitting consecutive containerised jobs in the same Slurm script (submitting jobs via a loop inside the script), because the system has not fully recovered memory from the previous job executed. We can minimise this by adding a sleep statement in the loop after each job submission.

Performance

It is possible to attain “bare metal” performance from applications and workflows running inside containers and in some cases achieving noticeable better performance (loading python packages). At LRZ, we have successfully executed an instance of machine learning training with TensorFlow on 768 nodes of SuperMUG-NG with multi-PetaFlop performance. To achieve this, it is important that the container is fully taking advantage of the underlying HPC systems hardware (compute and interconnect). This is achieved by installing and configuring the libraries and software for the high-speed interconnect inside the container, as well as ensuring that the system-optimised MPI library is being called at runtime. This was achieved by binding the module system directory inside the container and linking the systems MPI libraries at runtime.

Case Studies at LRZ

1. Enable ExaScale for EverYone (EASEY)

   Issues:

   - Containerised application crashed due to MPI issues - resolved by setting the libfabric parameters;
   - Poor performance due to running MPI over TCP - resolved by installing OmniPath software inside the container.

   Outcome:

   All the issues with stability and performance were no longer observed and the containerised application was able to execute successfully on 100's of nodes with 8000 MPI tasks.

2. Software Fuzzing

   Issues:

   - The applications IO pattern crashed the parallel file system:
     Resolution:
     - Switched to mounting a more performant directory of the parallel file server inside the container;
     - Switched to mounting the host systems RAM disk inside the container for storing temporary files.

   Outcome:

   Using the RAM disk of host system to store the millions of temporary files generated by the containerised application fixed the parallel file system crashes.

3. QuantEx (SuperMUC-NG)

   Issues:

   - Julia installs packages into ~/julia by default. Charliecloud maps the host ~ directory inside the container - resolved by changing the Julia package installation path and using the Docker environment instead of the host environment;
   - Profiling the Julia application using LIKWID inside the container - resolved by mounting the host module system inside the container.

   Outcome:

   Able to execute and profile the containerised QuantEX software on the HPC systems at LRZ.

Several papers have been produced describing the development and operational experience with Charliecloud at LRZ [58] , [59], [60], [61].

4.2.2 Singularity in production

Singularity has been available in CINECA, CESGA and Sigma2 clusters since 2017. In the context of PRACE-5IP, benchmark tests were performed both on parallel MPI and GPU workloads. The results were reported in scientific papers [62], [63] , [64]

In the context of PRACE-6IP, CINECA and Sigma2 activities were focused on maintaining Singularity as a production tool on all HPC clusters, making available the newer versions and

the guide for the users. Dissemination and training activities were also done by CINECA and Sigma2 staff. In what follows, the production deployment and usage procedures are described, mainly focusing on the actual cluster available. Some details about the dissemination and training activities are also reported.

Production deployment of Singularity in CINECA and usage procedures

During PRACE-6IP, the available HPC clusters in CINECA were MARCONI [65](in production since August 2017), MARCONI 100 [48] (in production since May 2020), GALILEO [49](available since March 2018 up to April 2021), GALILEO 100 [50] (in pre-production since July 2021) and D.G.X [51] (in production since January 2021).

Singularity has been installed as a production tool on all of them, providing a specific usage guide for the users [57]. Due to the different architectures and usage policy of the clusters, the production deployment and usage procedures of Singularity are different. In the rest of the section, on the basis of similar grouped clusters, the main installation and usage features of Singularity are reported.

Installation

On MARCONI, MARCONI 100 and GALILEO clusters Singularity is installed from source, and it is available to the users as a Linux module. All dependencies were installed by system administrators via CentOS/RHEL package, except for Go that was installed from source.

On GALILEO 100 and D.G.X. clusters, Singularity is installed by using the rpm package and it is available to the users as a Linux module.

The Singularity versions available are:

- GALILEO, MARCONI: version 3.6.1;
- MARCONI 100: version 3.7.0;
- GALILEO 100: version 3.8;
- D.G.X: version 3.5.3.

In the Linux module building, it does not set the SINGULARITY_HOME variable, to allow the automatic binding of the user HOME directory into the container. The SINGULARITY_CACHEDIR and SINGULARITY_TMPDIR are not set, in order to leave the users free to choose the preferred directories, if needed. Suggestions about the usage of such variables are provided in the user guide.

For Parallel MPI workflow [66], up to now we have supported only the Singularity “Hybrid model”, so in the Linux module there are no automatic bindings with the network and MPI libraries directories. To support the users, we have provided basic Singularity container images and recipes with all the network and MPI libraries needed: users can build their own container starting from such images/recipes. We plan to support the Singularity “Bind model”, by introducing the automatic bindings of the system network and MPI directories.
The Singularity fakeroot feature is not available on any clusters. Only on MARCONI 100, due to the fact that it is featured by a Power 9 architecture, we are providing to the users a virtual machine in which the “fakeroot” option is available, to allow them to build their own container image.

Usage procedures

By default, on all cluster users can run and/or execute their container images only by using a Slurm job scheduler. The images can be locally available on the clusters or stored in public and/or private hubs.

By design, users are not allowed to build their own container images on the cluster. An exception is done for MARCONI100, where a special virtual machine with the "fakeroot" feature is available.

Since Singularity is well integrated within Slurm, all the resources needed to run the container images must be selected by using the Slurm directives. Both Pure MPI, OpenMP and hybrid MPI+OpenMP jobs can be run.

For the MPI workflows, the users can run their application following both the Hybrid and the Bind model. In the second case they have to manually set the binding of all the needed network and MPI directories.
For GPU workflows, the option "--nv" must be added into the singularity command.

4.2.3 Unikernels

The experiments have been performed on two cluster of Grid'5000 [67]. The information given here is extracted from the Grid'5000 documentation [68]

- Nova is a cluster from the Lyon site of Grid'5000;
- Gros is a cluster from the Nancy site of Grid'5000.

The hyperthreading feature was disabled. We use HermitCore and HermiTux4from unikernels for our studies. Experiments are also performed on a Debian 10 distribution (Linux kernel version 4.19.0-14-amd64), to have a base reference. Due to unikernels limitation regarding OpenMP and compilers, we have been forced to compile specific executables for Linux and each unikernels:

- Linux executables are compiled with Clang/LLVM version 11, and use the LLVM OpenMP runtime version 11 (compiled with Clang v11);
- HermitCore executables are compiled with HermitCore's toolchain (GCC v6.3), and HermitCore'sOpenMP Intel runtime (v5.0) compiled with GCC v6.3;
- HermiTux executables are compiled with Clang/LLVM version 11, and linked with the LLVM OpenMPruntime version 11 (compiled with Clang v11) thanks to HermiTux' GCC wrapper.

It is important to note that we do not control the compiler used to compile HermitCore applications and which OpenMP runtime is used with HermitCore unikernel. As explained above, we did not manage to get the LLVM OpenMP runtime working with HermitCore. The results regarding HermitCore presented in the following section must be interpreted with caution.

Benchmarks Used

Experiments have been made with the Bots and Rodinias benchmarks.

Bots Benchmarks

The Bots benchmarks [70] are a set of benchmarks developed by the Barcelona Supercomputing Center. They are used for evaluating various OpenMP tasking implementations for some given problems. Here is a list of the benchmarks we used, extracted from the documentation of the Bots benchmarks:

- Alignment: Aligns sequences of proteins;
- FFT: Computes a Fast Fourier Transformation;
- Floorplan: Computes the optimal placement of cells in a floorplan;
- Health: Simulates a country's health system;
- NQueens: Finds solutions to the N Queens problem;
- Sort: Uses a mixture of sorting algorithms to sort a vector;
- SparseLU: Computes the LU factorisation of a sparse matrix;
- Strassen: Computes a matrix multiply with Strassen's method.

In Table 8, we check the boxes corresponding to the implementations we have at our disposal for the codes we used. The different implementations alter the tasks generation of the benchmarks:

- A tied implementation means that the task generation is limited. If this suffix is not present, it means that the task generation is unlimited;
- An if_clause implementation means that new tasks are generated when a condition is fulfilled. This condition is verified by an OpenMP directive;
- A manual implementation means that new tasks are generated when a condition is fulfilled, but this time, the condition is not checked by the OpenMP directive. It is done by a "manual" if statement in the sources;
- The for implementation generates tasks with an omp for directive. In this case, there can be multiple task generators;
- The single implementation means that there will be only a single task generator.

To compile the Bots benchmarks for HermitCore unikernel, a few modifications are performed on the sources of the Bots. Because the structure uts name is not defined in HermitCore, the program cannot compile with x86_64-hermit-gcc. This structure is returned by the uname() system call to describe the kernel name, release, and version. Every use of this structure in the benchmarks has been removed. It is not a critical functionality of the benchmarks so this deletion does not have any negative impact on the benchmarks. We also removed a call to the basename() function that was not supported by HermitCore. Again, it is not a critical feature of the benchmarks. Finally, a symbolic link has been created from /opt/hermit/lib/gcc/x86_64-hermit/6.3.0/include/memory.h [70] pointing to/opt/hermit/x86_64-hermit/include/hermit/memory.h. [71]
These modifications make the Bots benchmarks compatible with HermitCore.

Implementation	Alignment	FFT	Fib	Floorplan	Health	NQueens	Sort	SparseLU	Strassen
omp-tasks		✓	✓	✓	✓	✓	✓		✓
omp-tasks -tied		✓	✓	✓	✓	✓	✓		✓
omp-tasks -if_clause			✓	✓	✓	✓			✓
omp-tasks -if_clause-tied			✓	✓	✓	✓			✓
omp-tasks -manual			✓	✓	✓	✓			✓
omp-tasks -manual-tied			✓	✓	✓	✓			✓
for-omp -tasks	✓							✓	
for-omp -tasks-tied	✓							✓	
single-omp -tasks	✓							✓	
single-omp -tasks-tied	✓							✓	

Table 8: Problems solved by the Bots benchmarks, and their respective implementations.

Rodinias Benchmarks

The Rodinias benchmarks [73] are designed to evaluate different accelerators for compute-intensive applications: OpenMP, OpenCL, and CUDA. They are composed of already existing benchmarks, that have their unique behaviour, and come from many domains: Medical Imaging (Leukocyte, Heartwall...), Bioinformatics (MUMmerGPU), Fluid Dynamics (CFD Solver), Linear Algebra (for example LU Decomposition). Among the many benchmarks composing the Rodinias suite, only a subset were been selected for our study. In order to evaluate unikernels performance for HPC applications, we are interested in applications with long and intensive computation phases. Moreover, we only used the OpenMP version of the Rodinias benchmarks to study unikernels behaviour on multi-core CPUs. We observed a bug in HermiTux that caused some execution times to be negative. Further investigation showed that the bug occurred when using the gettimeofday() system call. This bug has been fixed. Sadly, two benchmarks still cause troubles. The bfs benchmark does not spawn threads correctly. The problem possibly comes from the unikernels, because the benchmarks always spawn the maximum number of threads it can for an execution on vanilla Linux (Debian 10), and only spawns one thread when executed with unikernels. The Kmeans benchmark source code contains a function definition that conflicts with another located inside HermitCore's kernel, causing errors at compile time. Because the Rodinias benchmarks do not share the common operations (such as time measuring, initialising etc.), making modification in these benchmarks in order to port them to unikernels is longer than in the Bots benchmarks. Due to a lack of time, we had to prioritise other topics. Finally, the benchmarks we managed to get working for both HermitCore and HermiTux are the following:

- lud (LU Decomposition) is a benchmark coming from the field of Linear Algebra. It is a benchmark decomposing a matrix as a product of matrices;
- LavaMD (LavaMD2) is a benchmark coming from the field of Molecular Dynamics. It calculates the position of particles in a 3D space considering the forces that apply between the particles.

4.2.4 Sarus

The experiments and performance measurements reported in this document were carried out on Piz Daint [41] a hybrid Cray XC50/XC40 system in production at the Swiss National Supercomputing Centre (CSCS) in Lugano, Switzerland. The system compute nodes are connected by the Cray Aries interconnect under a Dragonfly topology, notably providing users access to hybrid CPU-GPU nodes. Hybrid nodes are equipped with an Intel Xeon E5-2690v3 processor, 64 GB of RAM, and a single NVIDIA Tesla P100 GPU with 16 GB of memory. The software environment on Piz Daint at the time of the experiments is the Cray Linux Environment 6.0.UP07 (CLE 6.0) using Environment Modules to provide access to compilers, tools, and applications. The default versions for the NVIDIA CUDA and MPI software stacks are, respectively, CUDA version 9.1, and Cray MPT version 7.7.2.

We installed and configured Sarus on Piz Daint to use the native MPI and NVIDIA Container Toolkit hooks, and to mount the container images from a Lustre parallel filesystem.

Performance measurements

In experiments comparing native and container performance numbers, each data point presents the average and standard deviation of 50 runs, to produce statistically relevant results, unless otherwise noted. For a given application, all repetitions at each node count for both native and container execution were performed on the same allocated set of nodes.

4.3 Description of experiments and results

4.3.1 CUDA N-body

A fast n-body simulation is included as part of the CUDA Code Samples [42] The CUDA n-body sample code simulates the gravitational interaction and motion of a group of bodies. The code is written with CUDA and C and can make efficient use of multiple GPUs to calculate all-pairs gravitational interactions. More details of the implementation can be found in this article by Lars Nyland et al.: Fast N-Body Simulation with CUDA [42] .

We use this sample code to show that Sarus is able to leverage the NVIDIA Container Runtime hook in order to provide containers with native performance from NVIDIA GPUs present in the host system.

Test case

For this test case, we run the code with n=200,000 bodies using double-precision floating-point arithmetic on 1 Piz Daint compute node, featuring a single Tesla P100 GPU.

Running the container

We run the container using the Slurm Workload Manager and Sarus:

```
srun -Cgpu -N1 -t1 \
sarus run ethcscs/cudasamples:9.2 \
/usr/local/cuda/samples/bin/x86_64/linux/release/nbody -benchmark \
-fp64-numbodies=200000
```

Running the native application

We compiled and run the same code on Piz Daint using a similar Cuda Toolkit version (cudatoolkit/9.2).

Container image and Dockerfile

The container image ethcscs/cudasamples:9.2 (based on Nvidia cuda/9.2) used for this test case can be pulled from CSCS DockerHub [44] or be rebuilt with this Dockerfile:

```
FROM nvidia/cuda:9.2-devel

RUN apt-get update && apt-get install -y --no-install-recommends \
        cuda-samples-$CUDA_PKG_VERSION && \
    rm -rf /var/lib/apt/lists/*

RUN (cd /usr/local/cuda/samples/1_Utilities/bandwidthTest && make)
RUN (cd /usr/local/cuda/samples/1_Utilities/deviceQuery && make)
RUN (cd /usr/local/cuda/samples/1_Utilities/deviceQueryDrv && make)
RUN (cd /usr/local/cuda/samples/1_Utilities/p2pBandwidthLatencyTest && make)
RUN (cd /usr/local/cuda/samples/1_Utilities/topologyQuery && make)
RUN (cd /usr/local/cuda/samples/5_Simulations/nbody && make)

CMD ["/usr/local/cuda/samples/1_Utilities/deviceQuery/deviceQuery"]
```

Results

We report the gigaflops per second performance attained by the two applications in the following table.

	Average	Std. deviation
Native	3059.34	5.30
Container	3058.91	6.29

Table 9: gigaflops per second performance attained by the two applications

The results show that containers deployed with Sarus and the NVIDIA Container Runtime hook can achieve the same performance of the natively built CUDA application, both in terms of average value and variability.

4.3.2 OSU Micro-benchmarks

The OSU Micro Benchmarks (OMB) [45] are a widely used suite of benchmarks for measuring and evaluating the performance of MPI operations for point-to-point, multi-pair, and collective communications. These benchmarks are often used for comparing different MPI implementations and the underlying network interconnect.

We use OMB to show that Sarus is able to provide the same native MPI high performance to containerised applications when using the native MPICH hook. As indicated in the documentation for the hook, the only conditions required are:

- The MPI installed in the container image must comply with the requirements of the MPICH ABI Compatibility Initiative [46];
- The application in the container image must be dynamically linked with the MPI libraries.

Test case

The osu_alltoall benchmark measures the minimum, maximum and the average latency of the MPI_Alltoall blocking collective operation across N processes, for various message lengths, over a large number of iterations. In the default version, this benchmark reports the average latency for each message length up to 1MB. We run this benchmark from a minimum of 2 nodes up to 128 nodes, increasing the node count in powers of two.

Running the container

We run the container using the Slurm Workload Manager and Sarus.

```
srun -C gpu -N2 -t2 \
 sarus run --mpi ethcscs/osu-mb:5.3.2-mpich3.1.4 \
 ../collective/osu_alltoall
```

Running the native application

We compile the OSU Micro Benchmark suite natively using the Cray Programming Environment (PrgEnv-cray) and linking against the optimised Cray MPI (cray-mpich) libraries.

Container image and Dockerfile

The container image ethcscs/mpich:ub1804_cuda92_mpi314_osu (based on mpich/3.1.4) used for this test case can be pulled from CSCS DockerHub [42] or be rebuilt with the Dockerfile at [47].

Results

We collect latency values for 1kB, 32kB, 65kB and 1MB message sizes, computing averages and standard deviation. The results are displayed in the Figure 6:

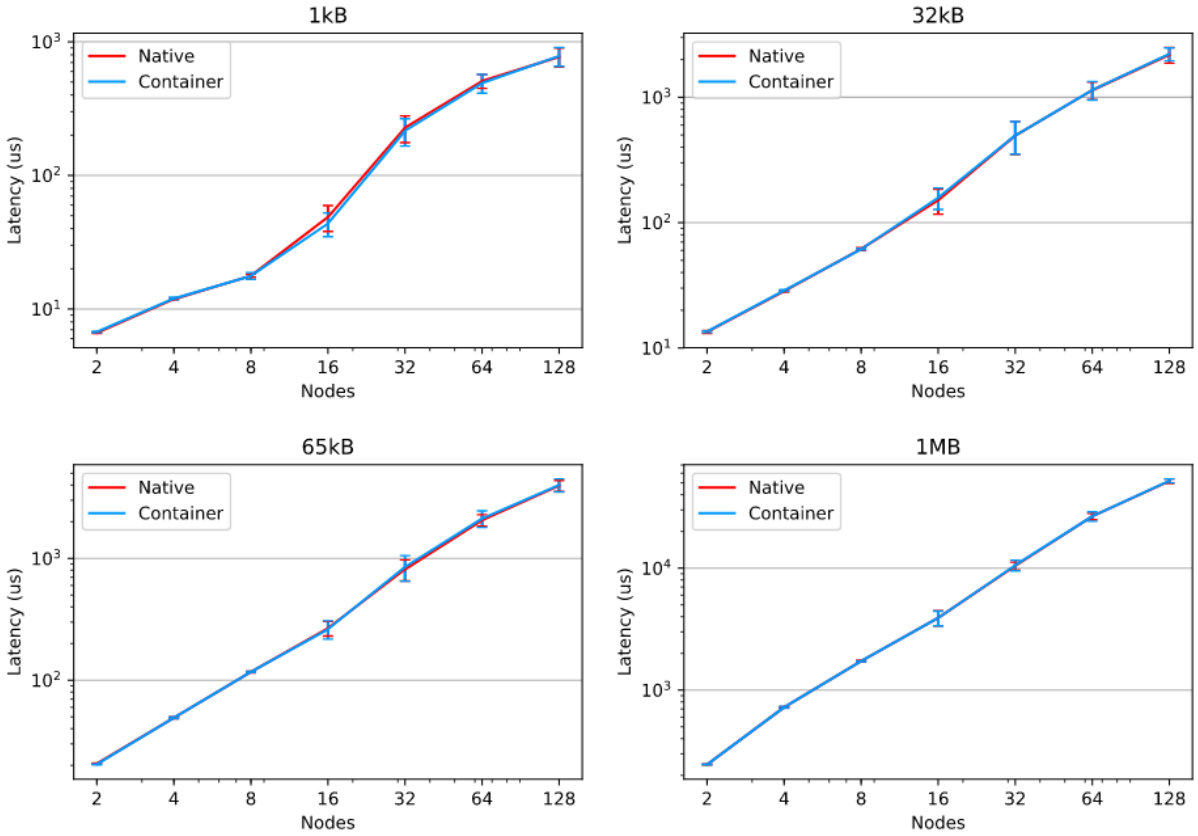

Figure 6: Results of the OSU All-to-all benchmark for the native MPI and MPICH 3.1.4 container. The MPI in the container is replaced at runtime by the native MPICH MPI hook used by Sarus.

We observe that the results from the container are very close to the native results, for both average values and variability, across the node counts and message sizes. The average value of the native benchmark for 1kB message size at 16 nodes is slightly higher than the one computed for the container benchmark.

It is worthy to note that the results of this benchmark are heavily influenced by the topology of the tested set of nodes, especially regarding their variability. This means that other tests using the same node counts may achieve significantly different results. It also implies that results at different node counts are only indicative and not directly relatable, since we did not allocate the same set of nodes for all node counts.

4.3.3 GROMACS

GROMACS [48] is a molecular dynamics package with an extensive array of modeling, simulation and analysis capabilities. While primarily developed for the simulation of biochemical molecules, its broad adoption includes reaserch fields such as non-biological chemistry, metadynamics and mesoscale physics. One of the key aspects characterising GROMACS is the strong focus on high performance and resource efficiency, making use of state-of-the-art algorithms and optimised low-level programming techniques for CPUs and GPUs.

Test case

As test case, we select the "3M" atom system from the HECBioSim [49] benchmark suite for Molecular Dynamics, a pair of hEGFR tetramers of 1IVO and 1NQL:

- Total number of atoms = 2,997,924
- Protein atoms = 86,996 Lipid atoms = 867,784 Water atoms = 2,041,230 Ions = 1,914

The simulation is carried out using single precision, 1 MPI process per node and 12 OpenMP threads per MPI process. We measured runtimes for 4, 8, 16, 32 and 64 compute nodes. The input file for the test case is available for download at [51] .

Running the container

Assuming that the benchmark.tpr input data is present in a host directory referred by the arbitrary environment variable $INPUT, we can run the container on 16 nodes as follows:

```
srun -C gpu -N16 sarus run --mpi \
    --mount=type=bind,src=${INPUT},dst=/gromacs-data \
    ethcscs/gromacs:2018.3-cuda9.2_mpich3.1.4 \
    /usr/local/gromacs/bin/mdrun_mpi -s /gromacs-data/benchmark.tpr -ntomp 12
```

Running the native application

CSCS provides and supports GROMACS on Piz Daint [52]. At the time of this experiment, the GROMACS/2018.3-CrayGNU-18.08-cuda-9.1 modulefile was loaded.

Container image and Dockerfile

The container image ethcscs/gromacs:2018.3-cuda9.2_mpich3.1.4 (based on cuda/9.2 and mpich/3.1) used for this test case can be pulled from CSCS DockerHub [51] or be rebuilt with this Dockerfile:

```
1  #!/bin/sh
2  FROM ethcscs/mpich:ub1804_cuda92_mpi314
3
4  # Install CMake (apt installs cmake/3.10.2, we want a more recent version)
5  RUN mkdir /usr/local/cmake \
6  && cd /usr/local/cmake \
7  && wget -q https://cmake.org/files/v3.12/cmake-3.12.4-Linux-x86_64.tar.gz \
8  && tar -xzf cmake-3.12.4-Linux-x86_64.tar.gz \
9  && mv cmake-3.12.4-Linux-x86_64 3.12.4 \
10 && rm cmake-3.12.4-Linux-x86_64.tar.gz \
11 && cd /
12
   ENV PATH=/usr/local/cmake/3.12.4/bin/:${PATH}
```

```
13
14  # Install GROMACS (apt installs gromacs/2018.1, we want a more recent version)
15  RUN wget -q http://ftp.gromacs.org/pub/gromacs/gromacs-2018.3.tar.gz \
16  && tar xf gromacs-2018.3.tar.gz \
17  && cd gromacs-2018.3 \
18  && mkdir build && cd build \
19  && cmake -DCMAKE_BUILD_TYPE=Release  \
20           -DGMX_BUILD_OWN_FFTW=ON -DREGRESSIONTEST_DOWNLOAD=ON \
21           -DGMX_MPI=on -DGMX_GPU=on -DGMX_SIMD=AVX2_256 \
22           -DGMX_BUILD_MDRUN_ONLY=on \
23           .. \
24  && make -j6 \
25  && make check \
26  && make install \
27  && cd ../.. \
28  && rm -fr gromacs-2018.3*
29
```

Results

We measure wall clock time (in seconds) and performance (in ns/day) as reported by the application logs. The speedup values are computed using the wall clock time averages for each data point, taking the native execution time at 4 nodes as baseline. The results of our experiments are illustrated in Figure 7:

Daint.

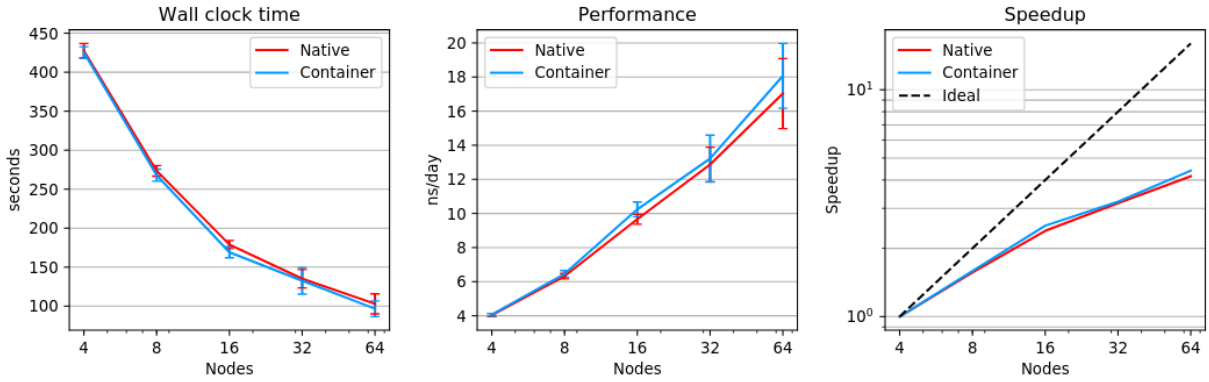

Figure 7: Comparison of wall clock execution time, performance, and speedup between native and Sarus-deployed container versions of GROMACS on Piz Daint

We observed the container application being up to 6% faster than the native implementation, with a small but consistent performance advantage and comparable standard deviations across the different node counts.

4.4 Dissemination and training activities

Below in this section are reported the main features of the dissemination and training activities conducted during PRACE-6IP

- Container Day - Bologna, 7-8 November 2019 [76] – the main Italian conference dedicated to container technologies, virtualisation and associated best practices.
- Containerisation in HPC, 2-days course - November 2020 [77] : we focused on an Introduction to containerisation, following by managing and executing Singularity and Docker containers. There were 116 participants, on line, with both theoretical and practical sessions.
- 17th Advanced School in Parallel Computing - Topic: Container tools for HPC, Singularity, Docker - March 2021 [78]: this was a PRACE School, focused on an Introduction to containerisation, followed by managing and executing Singularity and Docker containers. The last day was dedicated to Spack as a flexible package manager for HPC Software. There were 95 participants, it was an online event, with both theoretical and practical sessions.
- First PRACE training event for HPC container and Unikernels - 5-9 July 2021 [79]. Several platform were presented: Docker, Saarus, Singularity, Charliecloud, Unikernels. There were 47 participants, it was an online event, with both theoretical and practical sessions.

Use cases

23 cases have been supported during PRACE-6IP:

- Purpose of the software: Data analytics: 11 (55%), Virtualisation: 3 (15%), Deep learning: 6 (30%), Machine learning: 3 (15%), and Other: 6 (30%)
- Type of packaging: Virtual Machine: 5 (25%), Containers: 15 (75%).
- Support for parallelisation: Yes: 17 (85%), No: 3 (15%)
- Kind of parallelisation: Shared memory: 10 (50%), Distributed memory: 12 (60%)
- Approximate number of researchers using the software: Less than 5: 2 (10%), Between 5 and 20: 11 (55%), More than 20: 7 (35%)

4.5 Future of this service

Within a possible future PRACE project, this service will be mainly an operational support for containerised workload. The service will be providing container runtime platforms on Tier-0, Tier-1, and EuroHPC systems (LUMI, VEGA, and LEONARDO technical teams have already expressed interest in collaboration), in addition to container clusters (mainly kubernetes) on some PRACE sites. Containers can be either imported by users, or provided by PRACE repositories that store containerised HPC tools including general purpose and widely used MPI/GPU tools and containerised services. In addition, a front-end portal, which can be

replicated, with the capability of deploying docker and singularity tools on the fly. The service will include the following components:

- Tool repositories: Sharing of Docker and singularity containers in shared CVMFS and Docker registry repos. The repos can be mirrored on multiple sites to avoid centralisation;
- HPC Container runtime platforms: using runtime platforms that are connected to PRACE tool repos to pull tools;
- Frontend: Replicable web-based portal for pulling and running HPC containers (Galaxy);
- Virtualised services & Service orchestration;
- Container build templates: Github/Gitlab repository for Dockerfiles and Singularity recipes for MPI and GPU based containers;
- Documentation: Admin docs for installing and maintaining container platform runtimes and container repos mirrors & User docs for using different container platforms on different sites (including docs for MPI and GPU workloads).

To provide these components, the following tools will be used:

- Repository platforms: CVMFS server, Harbour Docker registry
  Categories of hosted tools:
  - HPC containers (Singularity): ML/DL tools, and other MPI/GPU tools that run as HPC jobs
  - Service containers (Docker): Jupyter notebook, RStudio, etc.
- Virtualized workload platforms: Docker, uDocker, Singularity, PCOCC, SARUS, uDocker, enroot
- Front-end interface: Galaxy portal
- Service orchestrators: Kubernetes, OpenShift, Docker Swarm

The service site providers of the service components:

- Virtualised tools: PRACE services/teams that package their tools in containers ;
- Runtime platform maintainers: CSCS (SARUS), CEA (PCOCC), CESGA (Singularity), IDRIS (Singularity), UiO/Sigma2 (Docker, Podman, enroot, uDocker) ;
- Virtualised workload support: UiO/Sigma2 (SAGA, FRAM, Betzy), CINECA (MARCONI, GALILEO, LEONARDO), CESGA (Fenis Terra II), CSC (LUMI (pre-exascale)), CEA (TBD), CSCS (Piz Diant) ;
- Github/Gitlab repository maintenance: UiO/Sigma2, CESGA, CINECA ;
- Container Repositories: UiO/Sigma2: CVMFS service and Harbour Docker registry ;
- Container service orchestrators: Sigma2 (Service Platform), CSC: Rahti cluster & Kubernetes on LUMI (pre-exascale) ;
- Galaxy Front-end: UiO/Sigma2: A portable Dockerised Galaxy portal.

The operational KPIs for the future service:

- Deployment simplicity: The ratio of container use cases that needed major modification to the original container file-system (from Docker hub) to all supported container use cases (should be minimum.);
- Portability: Containers that are built/used by one site, and needs modification to run on other sites to all PRACE containers (should be minimum);
- Performance: There should be no noticeable average performance degradation for containerised workloads compared to native workloads for similar applications;
- Stability: Failure incidents of container engines/platforms (should be minimum);

- Reproducibility: Results produced by using one or more containerised tools should be reproducible using the same container(s) ;
- Security: Security related incidents (e.g. users manage to get root privileges) should be minimum.

5 Service 4: Data Analytics

5.1 Description of the service

The partners involved in the Data Analytics service are IDRIS (Service leader), SURF, GRNET, EPCC, CINECA.

We identified at the beginning of the project, four objectives to achieve that are listed below:

- To transform this service into a regular service, meaning, to deploy a set of DL/ML services at different PRACE sites;
- To extend our connection to users to get more feedback that will be used to extend or enhance the services;
- To investigate more deeply the use of the graphical tools and the related issues they pose in an HPC environment;
- To investigate if new machine learning workflow tools can be relevant to orchestrate AI jobs (hyper-parameter search, ...) in HPC environments.

For each of these four goals, we describe in the following sections, the actions we put in place and the results we obtained.

5.2 Results achieved during the project

5.2.1 Service transformed to a regular service

The first goal of the working group was to transform the Data Analytics service into a regular service, consisting of a set of DL/ML services running at different PRACE sites. To reach this goal, we focused on the following actions:

- Identify the best set of service candidates to put in production. We used the results of the user surveys we launched to identify this set of tools. The overall user survey results are presented in the Connection to users section below and in the Annex part 7.2.
- Provide manual and automatic installation instructions. This documentation can be found in the Annex part 7.4.
- Provide user manuals. The manuals can be found in the Annex part 7.5.
- Several trainings were provided, ranging from site events to WP4 collaborations.

Given the experience of the Data Analytics group partners in running Data Analytics tasks, we proposed initially three core services to be supported in PRACE: Tensorflow, PyTorch and Horovod. Then, throughout the project, we paid attention to confirm, adjust and enhance the Data Analytics services offer by renewing periodic surveys.

In this way and given the survey results, the Data Analytics group recommends now to define:

- Tensorflow

- Keras
- PyTorch
- SckitLearn
- Horovod
- Jupyter

as the core services for PRACE, for Tiers-0 machines and for Tiers-1 machines running Data Analytics projects.

5.2.2 Connection to users

One of our concerns was also to extend our connection to users to get more feedback in order to enhance the PRACE Data Analytics services offer. However, when trying to contact ML/DL users, we realised that the use of these technologies may introduce some confidential issues that may not appear with other technologies. Indeed, some ML/DL users do not want to answer to user surveys in order to avoid telling which Data Analytics tools they use and even, on which topics they work before they have published their results. Given these facts, we decided to broaden the way of connecting users along three axes:

- We created a user questionnaire (see the Annex part Survey form 2021). It focuses on several aspects such as the Data Analytics tools used, the benefits obtained in running DA tasks on the PRACE infrastructure and the lack and difficulties that were faced;
- We used the EUSurvey tool to create it and to publish it on-line;
- We set up a collaboration with PRACE WP3 in order to benefit from the communication channel it offers to users. This way, more than 500 users were targeted from all PRACE partner countries in 2020 and 2021;
- We also contacted Tiers-0 and Tiers-1 PRACE WP7 representatives that are used to connecting with local users at each site, to get additional answers;
- We decided also to renew the survey periodically, in order to follow and to fit to the user practices.

We got around 70 answers to the questionnaire from all partner countries. The following figures describe the major results we got from these surveys. The survey form and additional figures can be found in the Annex part 7.2. We received wide distribution of answers by scientific domains as shown in Figure 8 below.

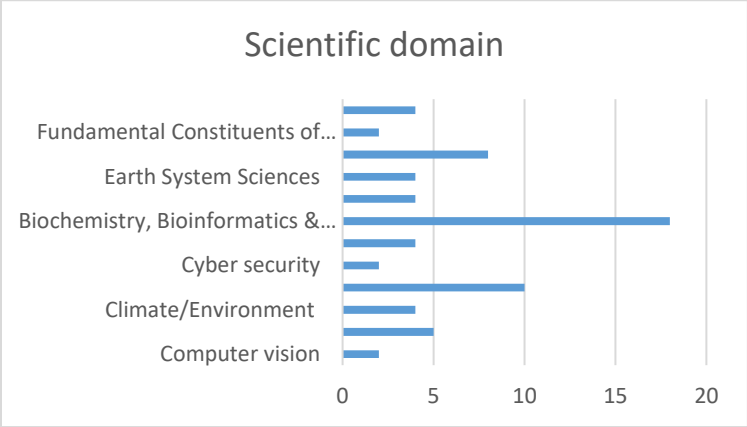

Figure 8: Distribution of answers by scientific domain

Figure 9 describes the various software stacks, ML/DL frameworks or libraries, used by the data analytics researchers. Even, Tensorflow, Keras, ScikitLearn and PyTorch appear to be widely used, one can also see that most of researchers use their own proprietary tools. Despite its very good performance, Horovod as a distributed training for Tensorflow and PyTorch, does not not seem to be widely used.

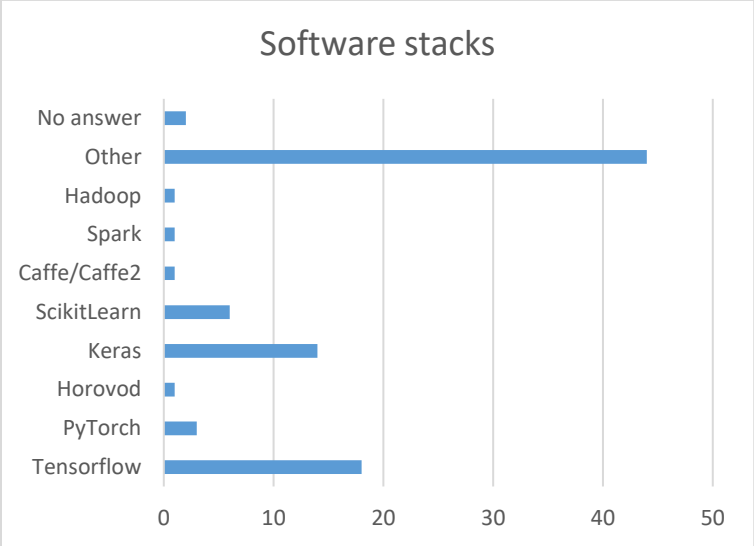

Figure 9: Software stacks, ML/DL frameworks and libraries

As shown by Figure 10 below, the most popular graphical tools used by ML/DL users are the CPU plotting package Matplotlib and Jupyter Notebooks.

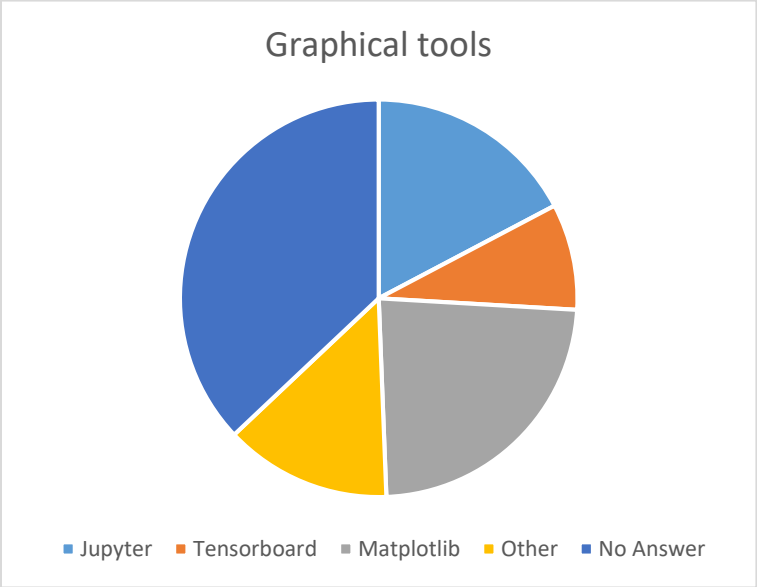

Figure 10 : Set of graphical tools used by Data Analytics users

Figure 11 below describes the dataset origin. It shows that even though most of the users have their dataset locally, a lot of users still have to transfer their data over the Internet which can be an issue for large datasets. In the case of public datasets, getting this data over the PRACE HP network and from a set of PRACE sites that make the most popular public datasets available to their users can solve this issue.

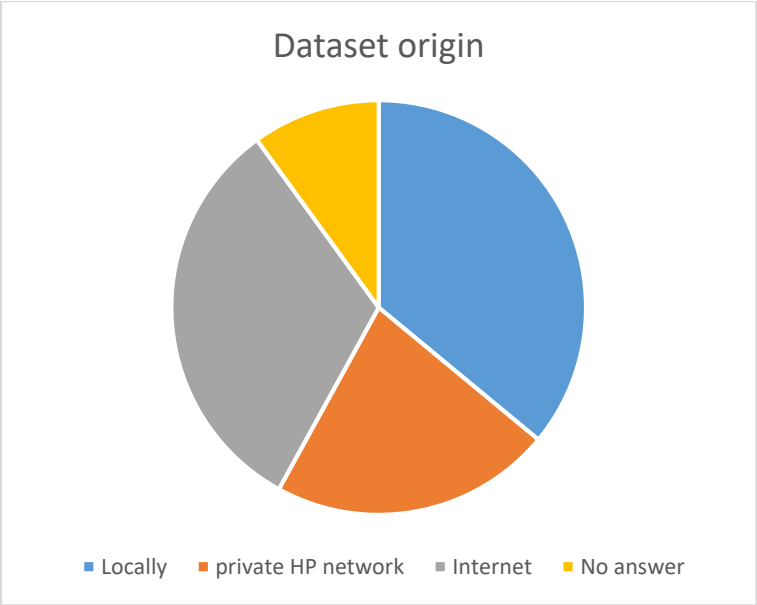

Figure 11: Dataset origin

Figure 12 shows the container usage for the Data Analytics tasks. From the set of container solutions available, two solutions stand out clearly: Singularity and Docker. It shows also, that most of the users did not answer to the question.

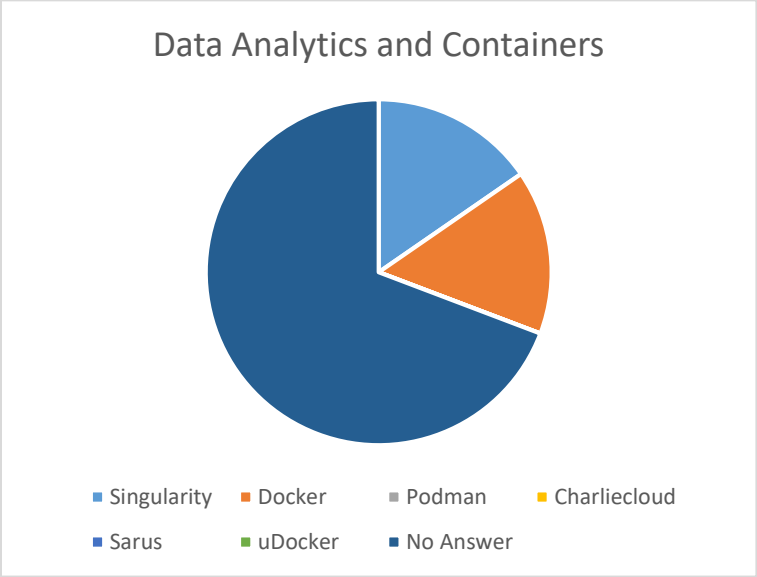

Figure 12: Container usage for Data Analytics

5.2.3 Graphical tools in HPC environment

The following two sections describe the custom environments that have been set up at IDRIS and at SURF around Notebooks and Hub from the Jupyter ecosystem.

5.2.3.1 Secure architecture to use and deploy secure Jupyter Notebooks and Tensorboard applications at IDRIS

The architecture described in Figure 13 below has been designed to bring a solution to hardware and software resources that are not accessible from outside the internal computer network. Also, security policies may require that only incoming SSH connections to front-ends are authorised and TCP forwarding are not allowed.

Jupyter Notebooks and Tensorboard are web applications accessible through the http protocol. In order to provide an authentication phase, a proxy server is used that supports SSL to encrypt communications between a service machine and the compute nodes. In this environment, Notebooks as well as Tensorboard that is run using a Notebook plugin can be launched in a secure way.

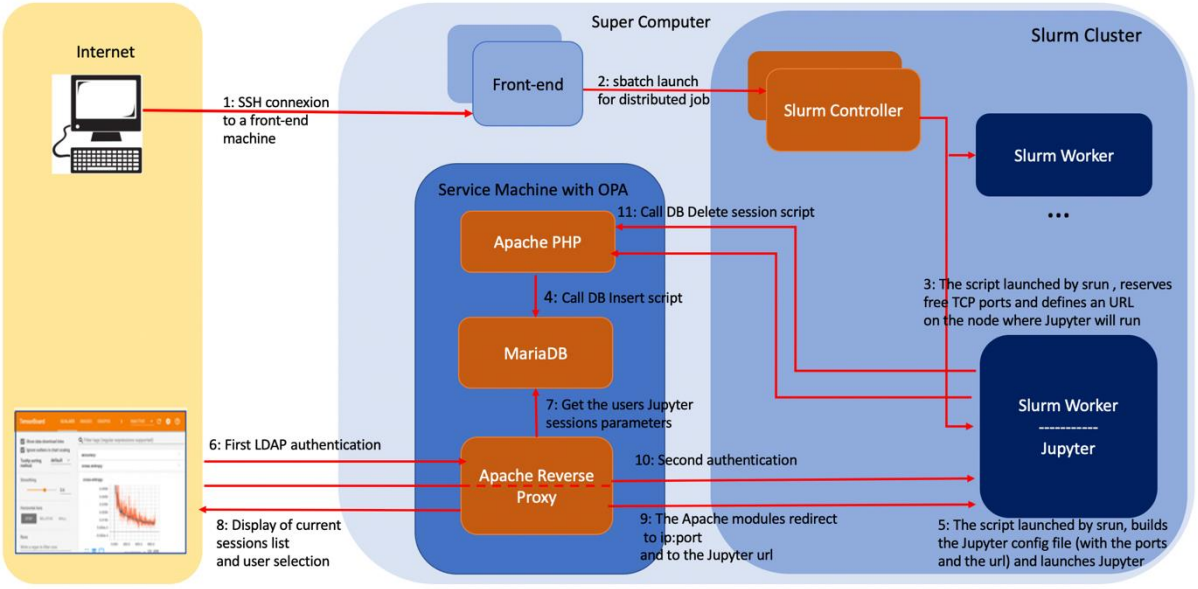

Figure 13: Secure architecture to use Jupyter Notebooks and Tensorboard.

The architecture operation is detailed in the Annex part 7.3.

5.2.3.2 JupyterHub deployment at SURF

At SURF, a Jupyter Notebook environment is offered on a SLURM cluster by using on a combination of the JupyterHub [80], the JupterHub batchspawner [81]and the JupyterHub wrapspawner [82]. Two user groups are targeted with this setup:

1. Traditional users of the cluster (i.e researchers), who use Jupyter Notebooks e.g. for code development or visualisation of results;
2. Course participants in e.g data science / machine learning courses that require a Python or R programming environment.

Multiple JupyterHub's are running on a service node inside the cluster. One JupyterHub instance is targeted at the first user group. In addition, each course gets its own instance of a JupyterHub. Running multiple JupyterHubs allows isolation of the different use cases, and allows different configurations for each JupyterHub. For example: the JupyterHub for researchers has options to spawn Jupyter Notebooks in allocations with either a single core CPU or single GPU, while the JupyterHub's for courses spawn Jupyter Notebooks with predefined

hardware (i.e either CPU or GPU, depending on the course) and in a SLURM reservation dedicated to that course.

Figure 14 shows the technical setup and an example workflow. In this example, the end user would go to https://my.hub.url/course2 and the Apache Reverse Proxy will forward this to the correct JupyterHub instance. Then, when the user starts the Jupyter Notebook server, the SLURM spawner of JupyterHub will submit a job to the SLURM controller. This job starts the Jupyter Notebook Server and will run at e.g https://my.hub.url/course2/user/user1. The JupyterHub for Course2 will make sure that this address gets forwarded to the right Notebook Server User 1, so that the end user is connected to the correct Jupyter Notebook Server instance.

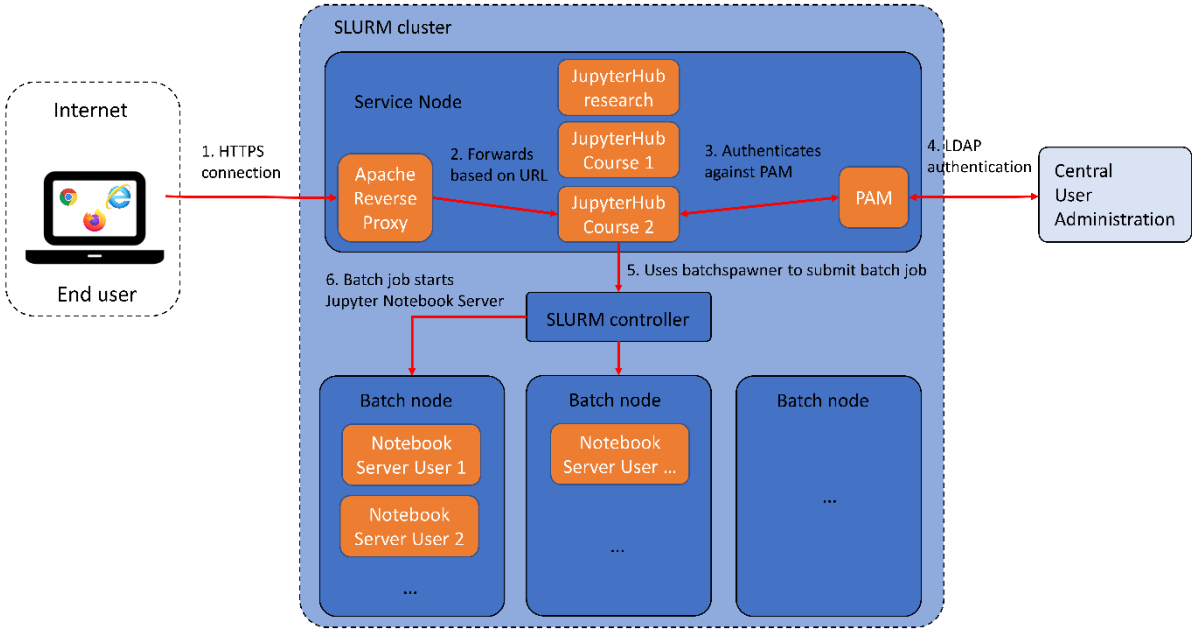

Figure 14: JupyterHub setup on a SLURM cluster at SURF.

5.2.4 Machine learning workflows tools

Machine learning projects require handling different versions of data, source code, hyperparameters, and environment configuration before machine learning models can be leveraged in production. We performed a quick evaluation of two tools able to manage this process: MLFlow and Polyaxon. We had also a look at Acumos. Despite its interest, it appeared not to be enough mature to be run in production in PRACE-6IP. The last tool is Open-CE which is run in production at CINECA.

5.2.4.1 MLFlow

Mlflow [83] is an open source platform developed by Databricks, to manage the machine learning lifecycle, including experimentation, reproducibility and deployment. It consists of three main components MLflow tracking, MLflow projects and MLflow models.
MLflow tracking is an API and a graphical interface (MLflow ui) that track parameters, performance indicators (metrics including user-defined metrics), code versions and output files. The graphical interface allows visualisation and comparison of results from different models. This feature is somewhat similar to Tensorboard but it can be used with various technologies

as soon as the source code is properly instrumented. MLflow tracking can be used with python, R, Java and REST and is designed to work with any machine learning library such as Tensorflow, Keras, PyTorch, ScikitLearn.

MLflow Projects helps in organising and in packaging the code. An MLflow Project describes the code dependencies and the way to run the code in order to produce reproductible results. MLflow supports several deployment modes and project environments: system environment, Conda environment, and Docker container environment on Kubernetes

An MLflow Model is a standard format for packaging machine learning models. MFflow uses the concept of flavors which are conventions that MLflow deployment tools can use to understand how to run a given model. This way, a model can be exported in one of these flavors to benefit from these deployment tools. The standard MFflow flavors include: python function, tensorflow, pytorch, keras, sklearn, spark, onnx. MLflow models can be deployed locally or can be packaged as Docker images.

5.2.4.2 Polyaxon

Polyaxon [84] is a commercial tool with jointly a Community Edition which is an open source platform able to manage the complete machine learning life cycle of large scale deep learning applications. It supports the most popular deep learning frameworks and machine learning libraries. Unlike MLflow, Polyaxon uses Kubernetes which makes it heavier to deploy but also provides additional functionalities. It relies on Kubernetes for managing the cluster resources (memory, CPU, GPU), for creating repeatable deployments and for scaling up and down.

Polyaxon focus on the reproducibility aspect of machine learning in order to allow it to easily recreate a workflow or an experiment to get the same results while being language and framework agnostic. This feature is based on a specification file (Polyaxonfile). Polyaxon supports hyperparameters search and optimization. It has its own dashboard to visualize and compare experiments based on results, hyperparameters, versions of training data and source code.

As MLflow, Polyaxon has a similar concept of *Model*. A Model is a format for packaging and managing machine learning models. It is easy to reproduce the way the model was built thanks to the information from the provenance of the initial experiment. In order to manage the experimentation and the automation process, Polyaxon relies on a set of runtime objects which are jobs, distributed jobs, services and DAG. These objects are used to execute a code or to run distributed jobs on a Kubernetes cluster. Services and DAG are used respectively to run graphical user interface such as Tensorboard or Jupyter Notebooks and to describe the workflows including operations and dependencies.

5.2.4.3 Acumos

Acumos AI [85] is a platform and open source framework that aims to build share and deploys AI applications. Acumos is part of the LF AI Foundation within The Linux Foundation that supports open source projects in machine learning and deep learning. Acumos supports the four stages of AI development as shown on Figure 15: the creation and On-board models, the model enhancement, the model sharing in marketplace and the execution in a target environment.

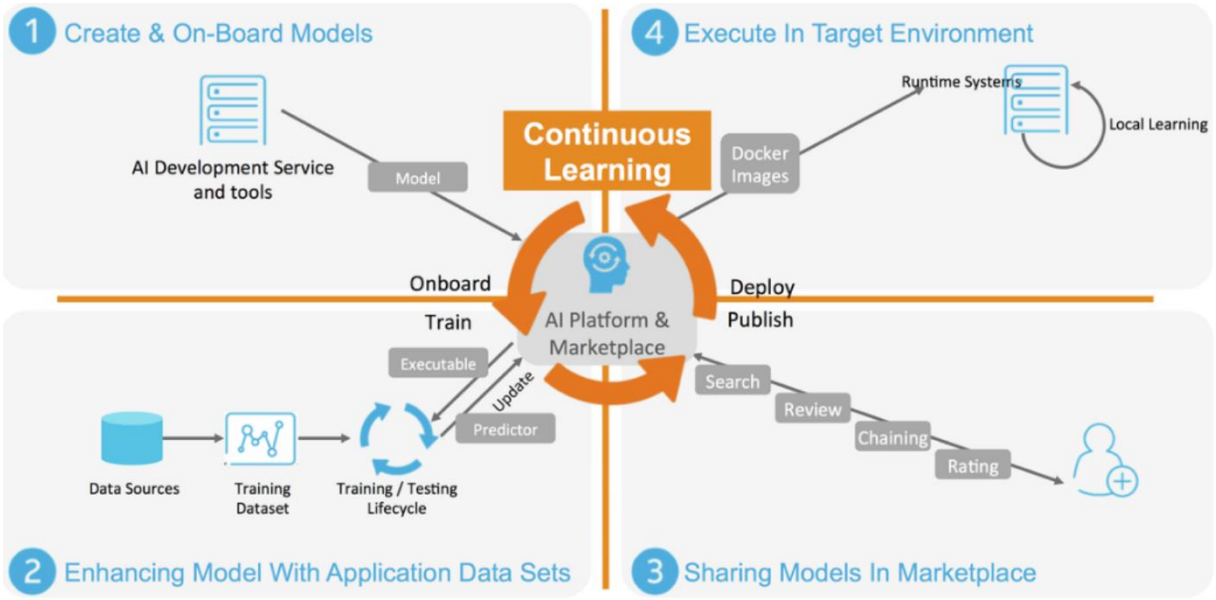

Figure 15: The four stages of AI Development supported by Acumos

The Acumos capabilities can be grouped into the following:

- Build models
  - Generate model packages for onboarding by CLI or Web;
  - Generate model microservice images based upon docker base images;
- Share models
  - Onboard models by CLI and Web;
  - Share and publish the models to company and public marketplaces;
- Deploy models
  - Download for local deployment under docker and kubernetes;
  - Deploy to public and private clouds (OpenStack);

The Acumos Marketplace is designed to make it easy to discover, explore, and use AI models. It allows the searching of models by keyword and filtering by category.

The Acumos Portal is designed to enable Modelers to easily on-board AI models. It is the place where modelers have the possibility to package them into reusable microservices to share then and publish the models. These microservices allow users to export AI packaged applications as containers to run in public clouds or private environments. Currently, Docker files are supported. Singularity containers are planned but not supported yet.

The Acumos Design Studio is used to chain together multiple models to create machine learning applications based on basic building blocks that are the individual models from the user community.

Acumos provides also several client libraries that allow modelers to push their models to the Acumos platform. Model On-Boarding instructions are available for several languages and environments: Java, Python, R, C++, Spark, ONNX and pre-dockerized models. Models sourced from toolkits such as ScikitLearn, TensorFlow or H2O can also be on-boarded. PyTorch is currently not supported.

The Onboarding functionality allows the user to choose to create or not the microservice. When a microservice generation is invoked, a docker image for the model is created and uploaded in

a Nexus docker repository. Models can currently be deployed into local or cloud Kubernetes clusters. The OpenStack deployment is on-going. The deployment to a batch scheduler environment like Slurm is currently not supported making it more difficult to adopt Acumos in an HPC environment.

Acumos data brokers provide capabilities for acquiring data from external sources, then using the data to train or tune models.

The Acumos release has been evaluated is Demeter. Even Acumos comes with 2 possible installation flows, it remains a heavy solution to deploy.

5.2.5 Open-CE

ML/DL frameworks, such as TensorFlow or PyTorch, can often be very difficult to install, also because these frameworks have a widely complex dependency chain; for example, TensorFlow relies on more than 80 open sources projects. In this task, different installation options, for example EasyBuild or Spack, were discussed bearing in mind the final goal of enabling researchers to use these tools rather than spending time installing or updating software packages.

Recently, Open-CE was established as a new community-driven set of ML/DL tools, enabling some of the best hardware with low-activation energy for users. Open-CE was built using IBM Watson Machine Learning Community Edition and OpenPOWER. This new version has a similar set of tools but can now be controlled and managed directly by the users who need the resource. It can include developers who want full control of all the versions of the tools and users who just want compiled binaries.

The Open-CE community is working to provide both. Open-CE GitHub page [86] focuses on providing feedstock to developers and groups while, for example, the Center for Genome Research and Biocomputing provides precompiled Conda packages [87]. The GitHub open-ce-builder [88] provides tools, based on conda-build, to build the conda package from the feedstock recipes. It is worth noting that machine learning users seem to be more familiar with tools like Conda with respect to Spack.

Open-CE provides a common dependencies environment for all supported packages, so that not only TensorFlow and PyTorch rely on the same version of their common dependencies but they could be even installed in the same environment and used in the same code.This has been made possible also since conda is multi architecture by default, with a real dependency software that is built in, while this is not the case for pip, that lacks a true dependency software. Let us note that while pip can be used inside a Conda environment the reverse does not work.

Open-Ce supports different variants of Python (3.7, 3.8, 3.9) and CUDA (10.2, 11.0, 11.2) that can be specified at build time as well as the MPI compiler can be chosen between openMPI or one already available on the system on which you are building on, with CUDA the only required dependency besides Conda. This seamless management of the whole dependency chain is one of the main advantages of the tool.

5.2.6 KPI

We identified at the beginning of the project, the following KPI as an interesting number to follow: *Number of scientific use cases that use the core services (Tensorflow, PyTorch and Horovod) available at PRACE HPC systems.*

This KPI was monitored since October 2020 with numbers checked every month. These numbers were fed with two user surveys through the WP3 or WP7 channels, one in 2020 and the other in 2021 (report is in Section 7). We set up a minimal requirement of three projects to support the core services defined.

From the set of answers we got, 18 projects used Tensorflow, 3 projects used PyTorch and 1 project used Horovod. It shows that Tensorflow as one of the first popular deep learning framework, has been widely adopted and remains very popular and intensely used among data analytics researchers. Despite its rich features and good performance, PyTorch usage appears to be still notably below Tensorflow. The use of Keras (fully integrated to Tensorflow since the release 2.0) meant that users did not feel the need to migrate their code from Tensorflow to PyTorch.

Despite its very good performance, Horovod as a distributed training for Tensorflow and PyTorch does not not seem to be widely used. This can be explained in different ways. First, maybe because most of users did not reach a sufficient complexity in their data analytics tasks that require distributed training. Secondly, maybe because users may use their own Tensorflow or PyTorch distributed training capabilities. Third, because it is now possible to have up to 8 GPUS on a single machine which pushes back the need to perform distributed training.

5.3 Future of this service

With the Data Analytics service being transformed into a regular service in PRACE, this means that the activity will be mainly related to the following ones:

- Monitoring the service at each site. So, the related Icinga reporters will have to be developed;
- Update of the PRACE Service Catalogue;
- Offer up-to-date tools to users by ensuring software maintenance at each site;
- Provide periodic surveys in order to check the service usage and follow the user requirements.

The Data Analytics task within some other EuroHPC JU funded activities could evolve in the following ways:

- Follow up of the regular service already started in PRACE (see tasks described above);
- Evaluation of new Data Analytics frameworks and libraries;
- User requirements follow up by providing periodic surveys;
- In EuroHPC JU, the Data Analytics task will have also to take into account the hardware evolution, both related to new GPUs generation and quantum computers.

Indeed, running machine and deep learnings tasks over quantum computers can drastically reduce computation times leading to an increasing usage of these technologies. Thus, an interesting topic will be to understand which frameworks, tools, applications and data set, environments can benefit the most of this new hardware technology.

6 Conclusions

In this deliverable, we present results obtained in Task 6.2 of Work Package 6 of the PRACE-6IP project. This task focused on four services that had potential to address some of the widely recognised needs in scientific computing and were already investigated within the preceding project PRACE-5IP: urgent computing, in-situ visualisation, the deployment of containers and full-virtualised tools into HPC infrastructures, and data analytics. The teams for each service were similar to the teams working on the service in PRACE-5IP, so the work was a smooth continuation of the previous project. The work of all teams was coordinated by UL FME.

Each team has done a long list of activities which have resulted in many test scripts, benchmark results, dissemination and training activities and are described in this document.

This deliverable has therefore four main sections – one for each service. Each section contains description of the service, a report of the work done within PRACE-6IP and final conclusion based on the results of the activities. We evaluated the KPIs that were defined at the beginning of the project and prepared a proposal how each service shall be continued beyond PRACE-6IP.

The main outcomes of Urgent computing (Service 1) was pilot implementation of two applications: Faster than real-time tsunami simulations (PTF/FTRT) and near real-time seismic scenarios (UCIS4EQ) and incorporating follow-up activities into a new EU project eFlows4HPC.

The main outcome of In-situ viasualisation service (Service 2) was succesfull installation of three in-situ frameworks (Catalyst, Damaris, Melissa) into the following seven HPC PRACE clusters: GALILEO, MARCONI, MARCONI100, Hawk, JEWELS, MareNostrum, and Jean-Zay. We also selected scientific CFD codes of relevant importance in the CFD community to be in-situ instrumented and developed a coupling interface or improved an existing one for the following codes: MIGALE, OpenFOAM, STREAmS, Code Saturne, ParFlow, and WRF. The first three codes have been coupled with Catalyst, the fourth with Damaris and the last two with Melissa.

The main outcomes of Service 3 (The deployment of containers and fully virtualised tools into HPC infrastructures) consist of several trainings and dissemination activities that were performed within PRACE and of support for 23 use cases where containers combined with parallelisation were a key-enabling technology.

The main outcome of the Data analytics service (Service 4) consists in the answers provided to the user survey that was run in two iterations and gave impotant feedback for the service partners. These results were the basis for the list of services, for which automatic installation instructions and user manuals were created. Additionally, the work on this service resulted in several trainings, ranging from site events to WP4 collaborations.

7 Annex to Service 4 – Data Analytics

7.1 Survey form 2021

This section describes the survey form for the Data Analytics service sent to users in 2021.

PRACE Data Analytics User feedback questionnaire 2021

Project details

Scientific domain of your project

- [ ] Speech recognition
- [ ] Computer vision
- [ ] Robotics
- [ ] Transport
- [ ] Health
- [ ] Telecom
- [ ] Climate/Environment
- [ ] Physics/Astrophysics
- [ ] Finance
- [ ] Cyber security
- [ ] Distribution/Commerce
- [ ] Energy
- [ ] Bank/Insurance
- [ ] Aeronautic/Space
- [ ] Biochemistry, Bioinformatics & Life Sciences
- [ ] Chemical Sciences & Materials
- [ ] Earth System Sciences
- [ ] Engineering
- [ ] Fundamental Constituents of Matter
- [ ] Universe Sciences
- [ ] Other

So, which scientific domain?

Project size (number of persons)

- ( ) 1 - 2 persons
- ( ) 3 - 5 persons
- ( ) 6 - 10 persons
- ( ) 11 - 20 persons
- ( ) more than 20 persons

How did you get access to the PRACE infrastructure?

- ( ) Preparatory Access
- ( ) Project Access
- ( ) Multi-year Access
- ( ) COVID-19
- ( ) DECI Call
- ( ) SHAPE Call
- ( ) Other

Project workflow and dataflow

Could you briefly describe your use case?

1

Compute resources

Consumption:

	Per job
Number of CPU/Cores	
Maximum number of GPUs	
Maximum number of nodes	
Memory per CPU/core	
Memory per GPU	
Elapsed time	

Dataset and storage resources

What storage capacity did you use (GB and inodes) for your project?

Which dataset type did you use?

- Private
- Public

Which public dataset did you use?

Where did you get your dataset?

- Locally (where the experiment ran)
- Throught a private high performance network
- Internet

Software tools

Which software stacks did you use?

- Tensorflow
- Pytorch

2

- [ ] Horovod
- [ ] Keras
- [ ] ScikitLearn
- [ ] Caffe/Caffe2
- [ ] Spark
- [ ] Hadoop
- [ ] Other

Which one?

Which release did you use?

Graphical tools

If you have used a graphical tool, could you specify which one?

- [ ] Jupyter
- [ ] Tensorboard
- [ ] Matplotlib
- [ ] Spark UI
- [ ] Other

Which one?

Containers environment

If you have used containers to run your experiment could you specify which technology?

- [ ] Singularity
- [ ] Docker
- [ ] Podman
- [ ] Charliecloud
- [ ] Sarus
- [ ] uDocker

Which release did you use?

3

Assessment of the PRACE infrastructure

Did you run your experiment on other infrastructure(s)? If yes, what were their characteristics?

What benefits did the PRACE infrastructure bring to your project?

From your point of view, which services of the PRACE infrastructure should be enhanced? in which way?

What new services would you like the infrastructure to bring?

Outcome

If the results were published, which journal, conference ...?

4

7.2 Additional survey results

This section presents some additional survey results. Figure 17 describes how users get access to the PRACE infrastructure.

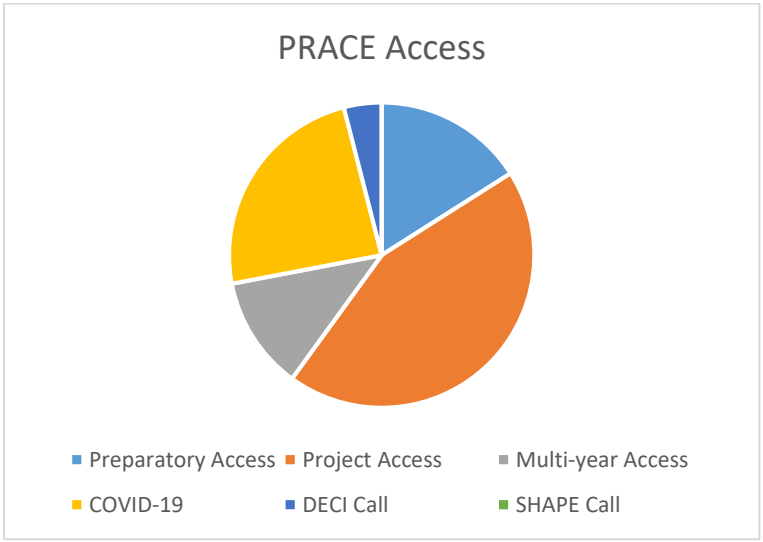

Figure 16 : Users access to the PRACE infrastructure

It shows that most of them use the project access way, followed by the COVID-19 projects, then the preparatory access way.
Figure 18 describes the project sizes to which the users belong.

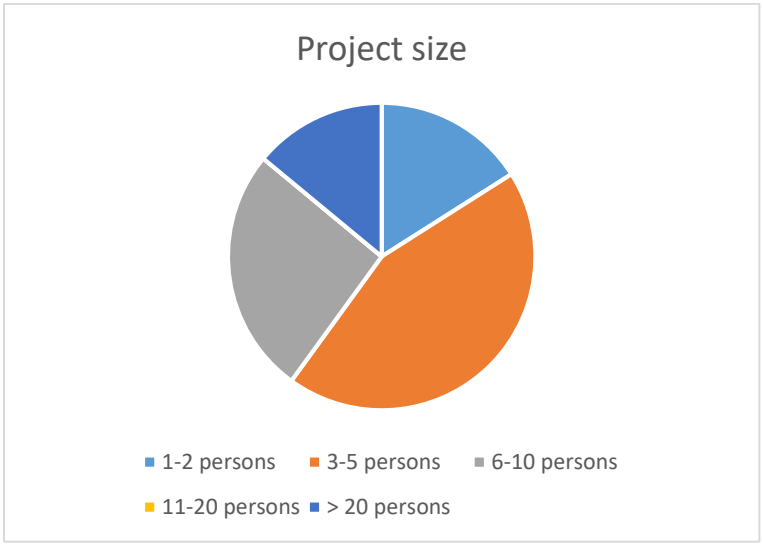

Figure 17 : Project sizes

Figure 19 describes the dataset type that is used if any.

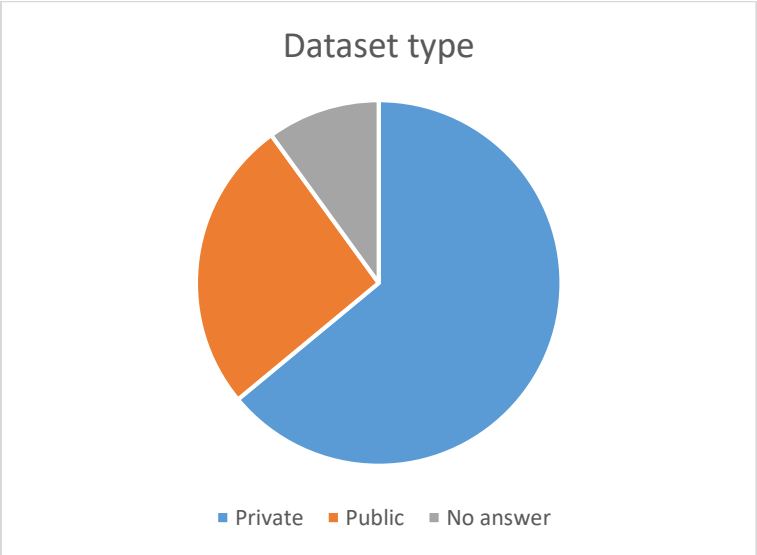

Figure 18 : Dataset type

7.3 Secure architecture Operation

The architecture described in Figure 13 is composed of the following components:

- A front-end
  - A user connects a front-end using ssh
  - The user requests computational resources and runs Slurm job using sbatch
- A Slurm cluster, composed of:
  - Slurm controllers that allocated resources to jobs on the Slurm workers and run the jobs on the Slurm workers
  - Slurm workers that run python process
    - The python process describes here is a web python application, that exposes data in http(s) and/or ws (WebSockets)
    - The python intended applications are currently Jupyter Notebook and Tensorboard
    - Tensorboard is launched using the jupyter-tensorboard plugin from a Jupyter Notebook
- A service machine that runs:
  - A http server (Apache as a reverse proxy)
  - A sql server (MariaDB to store sessions)
  - PHP pages to interface MariaDB
- Internet:
  - The user connects the front-end using ssh
  - The user has access through https to Jupyter Notebook or Jupyter Lab and to Tensorboard as a plugin of Jupyter

How this architecture works:

A user that plans to access to a web application proceeds as follows:

1) The user uses ssh to connect to a front-end (LDAP authentication backend)
2) The user submits a sbatch job to the Slurm controller which allocates one or more compute resources
3) On the allocated node(s), the command launched by srun:

   a. reserves free TCP ports that will be used by Jupyter
   b. defines an URL also used by Jupyter
4) The command launched by srun calls a PHP page running on the Apache server. The page will insert a record in MariaDB with the following information: user_id, job_id, nodename: port, url. The url parameter is built with nodename and port informations. Besides, the command generates a self-signed SSL certificate to encrypt communications between the service machine and the execution node.
5) The command launched by srun generates a random password to secure user access, launches a Jupyter server on a compute node allocated by slurm using the address https://nodename:port/url. These informations are printed out throught Linux standard output and stored into the Jupyter user's home folder.
6) The user connects to the public proxy as follows:
   a. https://proxyhost.domain
   b. User authenticates against the LDAP server
7) Using the LDAP user_id, Apache retrieves from MariaDB the list of running Jupyter sessions for this user
8) The list of sessions is displayed in the Web page and the user selects one
9) Apache performs the url mapping:
   a. The user sees https://proxyhost.domain/nodename_port/
   b. Apache maps this address with the cluster internal one: https://nodename:port
   c. Apache rewrites the urls in both directions using the mod_proxy and mod_rewrite Apache modules
10) For security reasons, a second password authentication is carried out by Jupyter. The password was generated in step 5.
11) When the command launched by srun ends, a PHP page is called to delete the related record in MariaDB. In case of epilog dysfunction, a scheduled SQL script purges the orphan and/or inactive sessions

7.4 Installation instructions

7.4.1 Anaconda

7.4.1.1 Anaconda installation

Anaconda is a Python distribution platform that is intensively used for data science and machine learning.
Anaconda includes :

- a package manager
- an environment manager
- a Python/R data science distribution
- large a collection of open-source packages

The conda utility makes it easy to manage multiple data environments (conda environment) that can be maintained and run separately.
To install Anaconda:

- Download the installer [89]
- Set the execution right, then run the installer script (for Linux) :
  - chmod u+x Anaconda3-2021.05-Linux-x86_64.sh
  - ./Anaconda3-2021.05-Linux-x86_64.sh

7.4.1.2 Anaconda verification

To check that anaconda has been properly installed, run :

```
conda list

# packages in environment at /xxxx/xxxx/anaconda3 :
#
# Name                    Version                   Build  Channel
_ipyw_jlab_nb_ext_conf    0.1.0                     py38_0
alabaster                 0.7.12             pyhd3eb1b0_0
anaconda                  2021.05                   py38_0
anaconda-client           1.7.2                     py38_0
anaconda-navigator        2.0.3                     py38_0
anaconda-project          0.9.1              pyhd3eb1b0_1
anyio                     2.2.0            py38hecd8cb5_1
appdirs                   1.4.4                      py_0
...
```

Anaconda provides more that 300 packages.

7.4.2 Tensorflow

7.4.2.1 Install TensorFlow precompiled binaries with pip

According to TensorFlow documentation the following operating systems are supported:

- Ubuntu 16.04 or later (64-bit)
- macOS 10.12.6 (Sierra) or later (64-bit) (no GPU support)
- Windows 7 or later (64-bit)
- Raspbian 9.0 or later

7.4.2.1.1 Prerequisites

- Python 3.5-3.7
- pip 19.0 or later
- GPU support requires a CUDA-enabled card

7.4.2.1.2 TensorFlow 2

The following TensorFlow 2 packages are available:

- tensorflow : Latest stable release with CPU and GPU support (Ubuntu and Windows)
- tf-nightly : Preview build (unstable). Ubuntu and Windows include GPU support.

7.4.2.1.3 Older TensorFlow versions

For TensorFlow 1.x, CPU and GPU packages are separate:

- tensorflow==1.15 : Release for CPU-only
- tensorflow-gpu==1.15 : Release with GPU support (Ubuntu and Windows)

7.4.2.2 Build and Install TensorFlow 2.x from Source

The following instructions have been tested on CentOS 6 and CentOS 7, but they are generic enough to work on other distributions as well.

7.4.2.2.1 Prerequisites

- Python 3.5 (we used 3.7.6, build instructions included below)
- Bazel 0.26.1
- GCC supporting c++14 (we used successfully GCC 6.5.0 and GCC 8.3.0. Do not use GCC 9.x because it is not supported by CUDA)
- Java 1.8.0
- CUDA (we used v. 10.1.168)
- NCCL (we used v. 2.4.7)
- cuDNN
- An MPI implementation (if you want to also install Horovod)

We also used

- Make 4.2
- Cmake 3.7.2
- Git 2.7.2 (Git 1.7.x that is by default installed on CentOS 6 will not work!)

Python, Bazel and TensorFlow have to be installed in the same isolated tree. From now on we assume the root of this tree is ${TENSORFLOWROOT}.

We also assume that ${COMPILER_GNUROOT} points to the installation directory of GCC. For example, if `which gcc` returns `/x/y/z/bin/gcc` then COMPILER_GNUROOT should be `/x/y/z`

7.4.2.2.2 Python

In case your OpenSSL is old (EL6) you need to install a recent OpenSSL (we used openssl-1.1.1d) before compiling Python:

```
tar zxf openssl-1.1.1d.tar.gz

cd openssl-1.1.1d

./Configure --prefix=${TENSORFLOWROOT} linux-x86_64; make; make install
```

Download Python's gzipped source tarball from https://www.python.org/downloads/source/ . Untar, configure and build:

```
tar zxf Python-3.7.6.tgz

cd Python-3.7.6

./configure --prefix=${TENSORFLOWROOT} --enable-shared --disable-ipv6 --
with-openssl=${TENSORFLOWROOT}

make

make install
```

After installing Python the following values should be prepended to the environment variables PYTHONPATH, PATH and LD_LIBRARY_PATH :

```
PYTHONPATH       ${TENSORFLOWROOT}/lib/python3.7/site-packages

PATH             ${TENSORFLOWROOT}/bin
```

```
LD_LIBRARY_PATH  ${TENSORFLOWROOT}/lib/python3.7/lib-
dynload:${TENSORFLOWROOT}/lib
```

In your ${TENSORFLOWROOT}/bin directory, create a 'python' symbolic link pointing to python3 :

```
cd ${TENSORFLOWROOT}/bin

ln -s python3 python
```

7.4.2.2.3 Python Modules

Install the following python modules :

```
pip3 install -U pip six numpy wheel setuptools mock 'future>=0.17.1'

pip3 install -U keras_applications --no-deps

pip3 install -U keras_preprocessing --no-deps
```

7.4.2.2.4 Bazel

Bazel is an open-source build and test tool that is used to build tensorflow. Bazel 0.26.1 is needed in order to build TensorFlow 2.x. You can find which is the latest Bazel release supported by TensorFlow in the TensorFlow code (check next section). Download Bazel 0.26.1 release from github :

https://github.com/bazelbuild/bazel/releases/download/0.26.1/bazel-0.26.1-dist.zip

Unzip in a directory:

```
mkdir bazel-0.26.1

cd bazel-0.26.1

unzip ../bazel-0.26.1-dist.zip
```

Build with:

```
env EXTRA_BAZEL_ARGS="--host_javabase=@local_jdk//:jdk" bash ./compile.sh
```

After build is complete, copy the bazel binary in the the $TENSORFLOWROOT/bin directory:

```
cp output/bazel ${TENSORFLOWROOT}/bin/
```

7.4.2.2.5 Download Tensorflow

Clone tensorflow git repository:

```
git clone https://github.com/tensorflow/tensorflow.git
```

Change directory into tensorflow, and check out the version you want to install. For this document we will use version 2.0.0, so

```
cd tensorflow

git checkout r2.0
```

In configure.py, search for the parameter "_TF_MAX_BAZEL_VERSION". This is the latest version of Bazel that is supported by this tensorflow version. For tensorflow 2.0.0, _TF_MAX_BAZEL_VERSION is 0.26.1 .

7.4.2.2.6 Build Tensorflow

Run the configure script:

```
./configure
```

You will need to specify a list of comma-separated CUDA compute capabilities of your GPU cards. Check the tables in

https://developer.nvidia.com/cuda-gpus#compute

You will also need to specify the NCCL version installed on your system. More info about NCCL:

https://developer.nvidia.com/nccl

You may also specify any optimization flags useful to your installation, for example for haswell CPUs:

```
-O3 -mavx2 -mfma -march=haswell -mtune=haswell
```

After the configure script is finished, make the following changes manually:

In tensorflow/tensoflow.bzl:

After:

```
    ctx.actions.run(
        executable = ctx.executable._swig,
        arguments = args,
```

add:

```
        use_default_shell_env=True,
```

In .tf_configure.bazelrc add the following line, adapted for your GCC version and path:

```
build --action_env GCC_HOST_COMPILER_PREFIX="/path/to/gcc/bin"
```

Set up C_INCLUDE_PATH :

```
export
C_INCLUDE_PATH=${TENSORFLOWROOT}/include:${COMPILER_GNUROOT}/include/c++/6.
5.0/x86_64-linux-gnu:${COMPILER_GNUROOT}/lib/gcc/x86_64-linux-
gnu/6.5.0/include-fixed:${COMPILER_GNUROOT}/lib/gcc/x86_64-linux-
gnu/6.5.0/include:${COMPILER_GNUROOT}/include/c++/6.5.0:${COMPILER_GNUROOT}
/include:/usr/include
```

Build the tensorflow pip package with:

```
bazel build  --config=opt //tensorflow/tools/pip_package:build_pip_package
```

and then

```
./bazel-bin/tensorflow/tools/pip_package/build_pip_package /tmp/tensorflow_pkg
```

7.4.2.2.7 Install tensorflow

Use pip to install the produced *whl package:

```
pip3 install tensorflow-2.0.1-cp37-cp37m-linux_x86_64.whl
```

7.4.2.2.8 Sample environment module for tensorflow 2.0.0

```
#%Module

# GRNET user environment

#

# Author                : ntell@grnet.gr

# Created               : Tue 15 Oct 2019 03:25:20 PM EEST

#

module-whatis "Enable usage for TENSORFLOW version 2.0.0 built with GNU Compilers 6.5.0"

proc ModulesHelp { } {

puts stderr "TensorFlow is an open source software library for numerical computation using
data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges
represent the multidimensional data arrays (tensors) communicated between them. The flexible
architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop,
server, or mobile device with a single API. TensorFlow was originally developed by researchers
and engineers working on the Google Brain Team within Google's Machine Intelligence research
organization for the purposes of conducting machine learning and deep neural networks
research, but the system is general enough to be applicable in a wide variety of other domains
as well."

}

prereq gnu/6.5.0

prereq java/1.8.0

prereq cuda/10.1.168

setenv TENSORFLOWROOT /apps/applications/tensorflow/2.0.0/gpu

prepend-path PATH $env(TENSORFLOWROOT)/bin

prepend-path LD_LIBRARY_PATH $env(TENSORFLOWROOT)/lib

prepend-path C_INCLUDE_PATH /usr/include

prepend-path C_INCLUDE_PATH /apps/compilers/gnu/6.5.0/include

prepend-path C_INCLUDE_PATH /apps/compilers/gnu/6.5.0/include/c++/6.5.0

prepend-path C_INCLUDE_PATH /apps/compilers/gnu/6.5.0/lib/gcc/x86_64-linux-gnu/6.5.0/include

prepend-path C_INCLUDE_PATH /apps/compilers/gnu/6.5.0/lib/gcc/x86_64-linux-gnu/6.5.0/include-
fixed

prepend-path C_INCLUDE_PATH /apps/compilers/gnu/6.5.0/include/c++/6.5.0/x86_64-linux-gnu

prepend-path C_INCLUDE_PATH $env(TENSORFLOWROOT)/include
```

7.4.3 Tensorboard

7.4.3.1 Tensorboard installation

Tensorboard is an interface designed for Tensorflow to visualize the learning process of a model. Whether you choose to install tensorflow from source code or using pip, tensorboard is installed at the same time.

You can check that the package is properly installed by running :

```
pip list | grep tensorboard
```

7.4.3.2 Tensorboard verification

To check that tensorboard has been properly installed run :

```
tensorboard --logdir=...
```

where logdir is the directory where tensorBoard will look to find tensorFlow event files coming from experiments.

Running this command will launch :

TensorBoard 2.5.0 at http://localhost:6006/

Please not that launching tensorboard such a way is not secure. Please refer to the secure web architecture describes in 5.2.3.1 to run tensorboard in a secure way in an HPC environment.

7.4.4 Keras

7.4.4.1 Keras and tf.keras

The official documentation is available at [90]

Keras is a python deep learning API that is widely used by researchers as it makes easier to perform deep learning tasks.

Keras needs a backend to build the network topology and to run the optimizers.

Originally, Keras default backend was Theano. It has moved progressively to a fully integrated Tensorflow default backend.

The migration was carried out in several phases :

- 2 independant libraries before TensorFlow v1.10.0
- since TensorFlow v1.10.0, a tf.keras submodule was introduced
- since TensorFlow v2.0, tf.keras is fully synchronized if you install Keras on your system, TensorFlow will also be installed.

The original keras package can be installed via pip :

```
pip install keras
```

Although, the keras package is still maintained with bug fixes, it is recommended to use the submodule tf.keras. The source code has to be changed in the following way :

```
from keras import ...
```

```
to :
```

```
from tensorflow.keras import ...
```

7.4.4.2 Keras verification

To check that keras has been properly installed (old Tensorflow releases), run the following python code :

```
python -c 'import keras; print(keras.__version__)'
```

The output should be similar to :

```
2.2.4
```

For recent Tensorflow release , run the following python code :

```
python -c 'import tensorflow.keras; print(tensorflow.keras.__version__)'
```

The output should be similar to :

```
2.4.0
```

7.4.5 PyTorch

7.4.5.1 Install PyTorch precompiled binaries with pip

According to PyTorch installation instructions, Linux distributions that use glibc >= v2.17 are supported, which include

- CentOS, minimum version 7.3-1611
- Debian, minimum version 8.0
- Ubuntu, minimum version 13.04

and others.

7.4.5.1.1 Prerequisites

- Python 3.6 or greater
- Anaconda or pip (we use pip in this document)
- CUDA, if you have a CUDA-capable system and want to use CUDA

7.4.5.1.2 Installation of PyTorch 1.4 module

If you wish to use CUDA v.10.1, run

```
pip install torch torchvision
```

If you wish to use CUDA v.9.2, run

```
pip install torch==1.4.0+cu92 torchvision==0.5.0+cu92 \
                  -f https://download.pytorch.org/whl/torch_stable.html
```

And finally if you would like to install PyTorch without using CUDA, run

```
pip install torch==1.4.0+cpu torchvision==0.5.0+cpu \
                  -f https://download.pytorch.org/whl/torch_stable.html
```

7.4.5.2 Build and Install PyTorch 1.4 from Source

The following instructions have been tested on CentOS 6 and CentOS 7, but they are generic enough to work on other distributions as well.

7.4.5.2.1 Prerequisites

- Python 3.5 (we used v. 3.7.6, build instructions included below)
- GCC supporting c++14 (we used successfully GCC 6.5.0 and GCC 8.3.0. Do not use GCC 9.x because it is not supported by CUDA)
- CUDA (we used v. 10.1.168)
- An MPI implementation (if you want to also install Horovod)

We also used

- Make 4.2
- Cmake 3.7.2
- Git 2.7.2

Python and PyTorch have to be installed in the same isolated tree. From now on we assume the root of this tree is ${PYTORCHROOT}.

We also assume that ${COMPILER_GNUROOT} points to the installation directory of GCC. For example, if `which gcc` returns `/x/y/z/bin/gcc` then COMPILER_GNUROOT should be `/x/y/z`

7.4.5.2.2 Python

In case your OpenSSL is old (EL6) you need to install a recent OpenSSL (we used openssl-1.1.1d) before compiling Python:

```
tar zxf openssl-1.1.1d.tar.gz

cd openssl-1.1.1d

./Configure --prefix=${PYTORCHROOT} linux-x86_64; make; make install
```

Download Python's gzipped source tarball from https://www.python.org/downloads/source/ . Untar, configure and build:

```
tar zxf Python-3.7.6.tgz

cd Python-3.7.6

./configure --prefix=${PYTORCHROOT} --enable-shared --disable-ipv6 --with-
openssl=${PYTORCHROOTT}

make

make install
```

Create an environment script (or environment modules script) that sets path, library path, pkg_info paths etc. with contents (sample is from environment-modules)

```
prereq gnu/8

prereq cuda/10.1.168

setenv PYTORCHROOT /path/topytorch/1.4.0

prepend-path PATH             $env(PYTORCHROOT)/bin

prepend-path LD_LIBRARY_PATH  $env(PYTORCHROOT)/lib

prepend-path LD_LIBRARY_PATH  $env(PYTORCHROOT)/lib64

prepend-path LD_LIBRARY_PATH  $env(PYTORCHROOT)/lib/engines-1.1

prepend-path LD_LIBRARY_PATH  $env(PYTORCHROOT)/lib/python3.7/lib-dynload

prepend-path PYTHONPATH       $env(PYTORCHROOT)/lib/python3.7/site-packages

prepend-path PKG_CONFIG_PATH  $env(PYTORCHROOT)/lib/pkgconfig

prepend-path PKG_CONFIG_PATH  $env(PYTORCHROOT)/lib64/pkgconfig
```

Update pip, install the absolutely required python packages.

```
pip3 install --upgrade pip

pip3 install mkl mkl-include numpy setuptools cffi pyyaml
```

Official PyTorch documentation recommends to also install `typing` and `cmake`

- DO NOT INSTALL typing if python is > 3.5.
- Do not install cmake if you have a relatively fresh system cmake installation.

7.4.5.2.3 OpenCV (Optional)

Many ML workloads include image manipulation. Install OpenCV (and dependencies : LibTIFF, libjpeg-turbo, jasper) in the same installation tree.

7.4.5.2.3.1 LibTIFF

Download tiff from http://www.simplesystems.org/libtiff/ . We have used version 4.0.9:

```
./configure --prefix=${PYTORCHROOT}

make; make install
```

7.4.5.2.3.2libjpeg-turbo

Download libjpeg-turbo from https://libjpeg-turbo.org/

We used version 1.4.1.

```
./configure --prefix=${PYTORCHROOT} --enable-shared
make; make install
```

7.4.5.2.3.3JasPer

Download JasPer from https://www.ece.uvic.ca/~frodo/jasper/

We used version 1.900.1

```
./configure  --prefix=${PYTORCHROOT}  --enable-shared  --without-x --
disable-opengl
```

7.4.5.2.3.4OpenCV

You need a recent cmake to compile OpenCV.

Download OpenCV from https://opencv.org.

Uncompress OpenCV source code and cd in the uncompressed source directory. Then:

```
mkdir build; cd build;
cmake \
        -DCMAKE_INSTALL_PREFIX=${PYTORCHROOT} \
        -DPYTHON3_EXECUTABLE=`which python3` \
        -DPYTHON_DEFAULT_EXECUTABLE=`which python3` \
        -DCMAKE_VERBOSE_MAKEFILE=true \
        -DWITH_CUDA=off \
        -DBUILD_opencv_python2=OFF \
        -DBUILD_opencv_python3=ON \
        -DTIFF_INCLUDE_DIR=${PYTORCHROOT}/include \
        -DTIFF_LIBRARY_RELEASE=${PYTORCHROOT}/lib/libtiff.so \
        -DWITH_PTHREADS_PF=off \
        -DWITH_OPENMP=on \
        -DJPEG_INCLUDE_DIR=${PYTORCHROOT}/include \
        -DJPEG_LIBRARY_RELEASE=${PYTORCHROOT}/lib/libjpeg.so \
        -DJASPER_INCLUDE_DIR=${PYTORCHROOT}/include \
        -DJASPER_LIBRARY_RELEASE=${PYTORCHROOT}/lib/libjasper.so \
        ..
make; make install
```

7.4.5.2.4 Download PyTorch

Download PyTorch with

```
git clone --recursive https://github.com/pytorch/pytorch
cd pytorch
git submodule sync
git submodule update --init --recursive
```

Check out the release you would like to build. For example for 1.4.0 :

```
git checkout v1.4.0
```

7.4.5.2.5 Build and install PyTorch

To build and install PyTorch run:

```
export CMAKE_PREFIX_PATH=${PYTORCHROOT}
export USE_CUDA=1
export USE_OPENCV=on
export TORCH_CUDA_ARCH_LIST="3.5"  # for K40, V100 is 7.0
export USE_GLOG=off
export LIB_MPFR=${COMPILER_GNUROOT}/lib/libmpfr.so
export LIBGMP=${COMPILER_GNUROOT}/lib/libgmp.so
export CMAKE_VERBOSE_MAKEFILE=on
export BUILD_NAMEDTENSOR=on
export USE_TBB=on
export MKL_INCLUDE_DIR=${PYTORCHROOT}/include

python3 setup.py install # build and install
```

7.4.5.2.6 Sample environment module for PyTorch 1.4.0

```
#%Module
# GRNET user environment
#
# Author                : ntell@grnet.gr
# Created               : Thu 23 Jan 2020 11:58:50 AM EET
#
module-whatis "Enable usage for Pytorch 1.4.0 with GNU Compilers 8 and CUDA
10.1"
proc ModulesHelp { } {
puts stderr ""
```

```
}

prereq gnu/8

prereq intelmpi/2018

prereq cuda/10.1.168

setenv PYTORCHROOT /apps/applications/pytorch/1.4.0

prepend-path PATH              $env(PYTORCHROOT)/bin

prepend-path LD_LIBRARY_PATH   $env(PYTORCHROOT)/lib

prepend-path LD_LIBRARY_PATH   $env(PYTORCHROOT)/lib64

prepend-path LD_LIBRARY_PATH   $env(PYTORCHROOT)/lib/engines-1.1

prepend-path LD_LIBRARY_PATH   $env(PYTORCHROOT)/lib/python3.7/lib-dynload

prepend-path PYTHONPATH        $env(PYTORCHROOT)/lib/python3.7/site-
packages

prepend-path PKG_CONFIG_PATH   $env(PYTORCHROOT)/lib/pkgconfig

prepend-path PKG_CONFIG_PATH   $env(PYTORCHROOT)/lib64/pkgconfig
```

7.4.6 Horovod

7.4.6.1 Install Horovod with pip

Horovod can be built and installed using pip. We recommend to have a separate Horovod module installed in each TensorFlow or PyTorch installation tree.

Horovod requires an MPI implementation, so before installing/building Horovod please load the MPI module of your choice (we used Intel MPI 2018).

7.4.6.1.1 TensorFlow

After loading the TensorFlow environment module, run

```
HOROVOD_WITH_MPI=1 HOROVOD_GPU_ALLREDUCE=NCCL HOROVOD_GPU_BROADCAST=NCCL
HOROVOD_WITH_TENSORFLOW=1 HOROVOD_WITHOUT_PYTORCH=1 pip install --no-cache-
dir horovod
```

If the build script fails to find your CUDA installation, add also

```
HOROVOD_CUDA_HOME=/path/to/your/cuda
```

Note: In our installation we also had to add an "-mfma" compiler flag, to get around this: "error: inlining failed in call to always_inline ‘__m256d _mm256_fmadd_pd(__m256d, __m256d, __m256d)’: target specific option mismatch"

```
HOROVOD_BUILD_ARCH_FLAGS="-mfma"
```

7.4.6.2 PyTorch

The procedure is similar with TensorFlow, but now we use HOROVOD_WITHOUT_TENSORFLOW=1 and HOROVOD_WITH_PYTORCH=1.

So after loading the PyTorch environment module, run

```
HOROVOD_WITH_MPI=1 HOROVOD_GPU_ALLREDUCE=NCCL HOROVOD_GPU_BROADCAST=NCCL
HOROVOD_WITHOUT_TENSORFLOW=1 HOROVOD_WITH_PYTORCH=1 pip install --no-cache-
dir horovod
```

If the build script fails to find your CUDA installation, add also

```
HOROVOD_CUDA_HOME=/path/to/your/cuda
```

We also had to add the '-mfma' flag (you may not need it depending on your CPU/compiler).

7.4.7 ScikitLearn

7.4.7.1 Install ScikitLearn binaries with pip

Scikitlearn is a set of python modules for machine learning and data mining.

The official documentation is available at [91]

Scikitlearn is shipped with Anaconda, see 7.4.1.1. Alternatively, it can be installed with miniconda or pip.

The following instructions describe the use of conda and conda environment:

Install conda from Miniconda.

Then run :

```
conda create -n aiwork python=3.8

conda activate aiwork

conda install scikit-learn
```

7.4.7.2 ScikitLearn verification

To check that scikitlearn has been properly installed, run the following Python code:

```
python -c "import sklearn; sklearn.show_versions
```

The output should be similar to :

```
System:
python: 3.8.10 | packaged by conda-forge | (default, May 10 2021, 22:58:09)
[Clang 11.1.0 ]
executable: /Users/XXX/anaconda3/envs/aiwork/bin/python
machine:
```

Python dependencies:

```
pip: 21.1.2
setuptools: 49.6.0.post20210108
sklearn: 0.24.2
numpy: 1.21.0
scipy: 1.6.3
Cython: None
pandas: None
matplotlib: None
```

```
joblib: 1.0.1
threadpoolctl: 2.1.0
Built with OpenMP: True
```

7.4.8 Jupyter Notebook

7.4.8.1 Install Jupyter Notebook binaries with pip

The official documentation is available at [92]

The Jupyter Notebook is a web-based application that is used for interactive computing. It allows to create documents that contain different contents such as code, text, mathematical equations and graphics.

Jupyter Notebook is shipped with Anaconda see 7.4.1.1. Alternatively, it can be installed with miniconda or pip.

The following instructions describe the use of conda and conda environment:

Install conda from Miniconda, see [93]

Then run :

```
conda create -n jnwork python=3.8

conda activate jnwork

conda install notebook
or pip install notebook
```

7.4.8.2 Jupyter Notebook verification

To check that notebook has been properly installed, run :

```
jupyter notebook
```

Running this command will launch a notebook server, for example :

```
Jupyter Notebook 6.3.0 is running at:
http://localhost:8888/?token=5a0e01f7bfb9b20201ad2f5a5a9279f34861cd9dedf74c
08
```

Please not that launching Jupyter Notebook such a way is not secure. Please refer to the secure web architecture describes at 5.2.3.1 to run jupyter notebook in a secure way, in an HPC environment.

7.4.9 Open-CE

7.4.9.1 Open-ce 1.1.3 building

The following instructions have been tested on

- Power9 ppc64le with Red Hat 8.1 and cuda 11.0
- Intel x86_64 with CentOS 7 and cuda 10.2

but they are generic enough to work on other distributions as well.

7.4.9.2 Open-ce prerequisites

- conda >= 3.8.3 (it can either be installed through Anaconda or Miniconda)
- conda-build == 3.20.5 (it can be installed with the command: conda install conda-build *)
- python >= 3.6
- docker >= 1.13 (optional, only required to create docker images)
- Cuda 10.2 or 11.0 (only required for building NVIDIA GPU version)
    - CUDA_HOME environment variable to be set to the location of the CUDA installation
    - TensorRT must be downloaded from [94] and saved in a directory called local_files adjacent to the open-ce repository. The tar.gz file must match version 7.0.0.11 or 7.2.* for cuda 10.2 or cuda 11.0 respectively.

7.4.9.3 Building a collection of packages

The following commands will use the opence-env.yaml Open-CE environment file to build all the Open-CE packages for Python 3.6 (the default), including CUDA builds and cpu-only builds (also the default). The commands should be run from within the same directory that contains local_files.

```
# Clone Open-CE from GitHub
git clone https://github.com/open-ce/open-ce.git --branch open-ce-v1.1.3
# Build packages
./open-ce/open-ce/open-ce build env open-ce/envs/opence-env.yaml
```

A complete list of the open-ce command options can be get using the –h option or found in the git repository [16] and allow to specify different settings such as:

- --output_folder OUTPUT_FOLDER: path where built conda packages will be saved (default: condabuild)
- --python_versions PYTHON_VERSIONS: Comma delimited list of python versions to build for, such as 3.7,3.8 (default: 3.6)
- --build_types BUILD_TYPES: Comma delimited list of build types, such as cpu or cuda (default: cpu,cuda)
- -mpi_types MPI_TYPES: Comma delimited list of mpi types, such as openmpi or system (default: openmpi)
- --cuda_versions CUDA_VERSIONS: CUDA version to build for, such as 10.2 or 11.0 (default: 10.2)

For example, the following command will build all the Open-CE packages for Python 3.8, including only cuda 11 builds, included openmpi and it will save the packages in $HOME/channel:

```
./open-ce/open-ce/open-ce build env --python_versions 3.8 --build_type=cuda
--cuda_versions 11.0 --mpi_type=openmpi --output_folder $HOME/channel open-
ce/envs/opence-env.yaml
```

7.4.9.4 Building a single package

The open-ce build env can be used also to build a single package such as tensorflow or pytorch (or any other in the open-ce/envs subdirectory), with dependencies automatically handled. For example, a build for pytorch may look like this:

```
 ./open-ce/open-ce/open-ce build env --python_versions 3.8 --
build_type=cuda --cuda_versions 10.2 --mpi_type=openmpi --output_folder
$HOME/channel open-ce/envs/pytorch-env.yaml
```

7.4.9.5 Installing packages

After performing a build, a local conda channel will be created. By default, this will be within a folder called condabuild but it can be changed using the --output_folder option. Packages can be installed within a conda environment from this local channel. For example:

```
conda install -c $HOME/channel pytorch
```

The open-ce build env script generate also a conda environment file which can be used to generate a conda environment with the built packages installed in it. For example:

```
conda env create ?f $HOME/chanel/opence-conda-env-py3.8-cuda-openmpi-
10.2.yaml
```

7.4.9.6 Environment set up

The straightway to setup the environment is to use conda activate. Nevertheless, a module file can be set up:

```
#%Module

# module: open-ce-1.1.3
# author: Marco Rorro
# creation date: 20210315 16:06:18

module-whatis "The Open-CE conda environment, includes (for example)
Tensorflow, Pytorch, XGBoost, and other related packages and dependencies."

prereq anaconda prereq cuda/11.0

set OPEN_CE_ENV "$rootdir_name/opence-conda-env-py3.8-cuda-openmpi-
11.0/opence"
set PYTHONPATH "$OPEN_CE_ENV/lib/python3.8/site-packages"
prepend-path PATH "$OPEN_CE_ENV/bin"
prepend-path LD_LIBRARY_PATH
"$OPEN_CE_ENV/lib:$OPEN_CE_ENV/lib64:$PYTHONPATH/nvidia/dali:$PYTHONPATH/te
nsorflow"
prepend-path PYTHONPATH "$PYTHONPATH"
setenv TF_INCLUDE_DIR "$PYTHONPATH/tensorflow/include"
setenv TF_LIBRARY_DIR "$PYTHONPATH/tensorflow"
```

7.5 User manuals

7.5.1 Tensorflow

7.5.1.1 Introduction

TensorFlow is an open source software library for numerical computation using data flow graphs. The graph nodes represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture enables you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code. TensorFlow also includes TensorBoard, a data visualization toolkit.

TensorFlow was originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence Research organization for the purposes of conducting machine learning and deep neural networks research. The system is general enough to be applicable in a wide variety of other domains, as well.

More information about TensorFlow can be found on the TensorFlow webpage.

7.5.1.2 TensorFlow Module

Usually on a supercomputer system multiple versions of TensorFlow are available. Each of these versions requires different environment settings, for example different values of the $PATH and $LD_LIBRARY_PATH environment variables. The Environment Modules package is used to dynamically modify the user's environment according to the TensorFlow (or other software) version that has to be used.

7.5.1.3 Use with Horovod

Horovod is included in the PRACE TensorFlow modules. So it is possible to use Horovod after loading the TensorFlow module, following the instructions in the TensorFlow documentation.

7.5.1.4 Module Usage

To list all available modules on a system, use:

```
module avail
```

To list all available versions of TensorFlow, use:

```
module avail tensorflow
```

Example:

```
# module avail tensorflow
--------------------------- /apps/modulefiles/applications ----------------------
tensorflow/1.10.1gpu  tensorflow/1.5        tensorflow/1.8gpu     tensorflow/2.0.0
tensorflow/1.12.0gpu  tensorflow/1.5gpu     tensorflow/1.9        tensorflow/2.1.0
tensorflow/1.14.0     tensorflow/1.8        tensorflow/1.9gpu     tensorflow/2.2.0
```

To check module dependencies and the environment changes that will happen if a module is loaded, use '`module show`' . Example

```
# module show tensorflow/2.1.0

-------------------------------------------------------------------

/apps/modulefiles/applications/tensorflow/2.1.0:


module-whatis    Enable usage for TENSORFLOW version 2.1.0 built with GNU Compilers
6

prereq   gnu/6

prereq   java/1.8.0

prereq   cuda/10.1.168

prereq   intel/18

prereq   intelmpi/2018

setenv           TENSORFLOWROOT /apps/applications/tensorflow/2.1.0/gpu

prepend-path     PATH /apps/applications/tensorflow/2.1.0/gpu/bin

prepend-path     LD_LIBRARY_PATH /apps/applications/tensorflow/2.1.0/gpu/lib

prepend-path     LD_LIBRARY_PATH /apps/applications/tensorflow/2.1.0/gpu/lib64

prepend-path     LD_LIBRARY_PATH
/apps/applications/tensorflow/2.1.0/gpu/lib64/engines-1.1

prepend-path     LD_LIBRARY_PATH
/apps/applications/tensorflow/2.1.0/gpu/lib/python3.7/lib-dynload

prepend-path     C_INCLUDE_PATH /usr/include

prepend-path     C_INCLUDE_PATH /apps/compilers/gnu/6.5.0/include

prepend-path     C_INCLUDE_PATH /apps/compilers/gnu/6.5.0/include/c++/6.5.0

prepend-path     C_INCLUDE_PATH /apps/compilers/gnu/6.5.0/lib/gcc/x86_64-linux-
gnu/6.5.0/include

prepend-path     C_INCLUDE_PATH /apps/compilers/gnu/6.5.0/lib/gcc/x86_64-linux-
gnu/6.5.0/include-fixed

prepend-path     C_INCLUDE_PATH /apps/compilers/gnu/6.5.0/include/c++/6.5.0/x86_64-
linux-gnu

prepend-path     C_INCLUDE_PATH /apps/applications/tensorflow/2.1.0/gpu/include

prepend-path     PKG_CONFIG_PATH
/apps/applications/tensorflow/2.1.0/gpu/lib64/pkgconfig

-------------------------------------------------------------------
```

To load a TensorFlow module, run '`module load`', with the version you wish to load and its dependencies. For example, if we would like to load version 2.1.0 :

```
module load gnu/6 java/1.8.0 cuda/10.1.168 intel/18 intelmpi/2018 tensorflow/2.1.0
```

To check loaded modules use '`module list`' :

```
# module list

Currently Loaded Modulefiles:
```

```
  1) gnu/6              2) java/1.8.0         3) cuda/10.1.168      4) intel/18
5) intelmpi/2018      6) tensorflow/2.1.0
```

To clean your environment from all loaded modules use '`module purge`' :

```
# module purge

# module list

No Modulefiles Currently Loaded.
```

7.5.1.5 Job Submission

To run a job, a job script has to submitted to the system's Workload Manager, usually SLURM. An example of a SLURM batch script is provided below. However most details, like partition, account, tasks per node, will be different on each PRACE system. Please consult the system's documentation and support for the correct values.

```
#!/bin/bash -l


#-----------------------------------------------------------------
# GPU job on 4 nodes ,
# with 2 gpus per node and 20 threads per MPI task.
#-----------------------------------------------------------------


#SBATCH --job-name=gpujob # Job name
#SBATCH --output=gpujob.%j.out # Stdout (%j expands to jobId)
#SBATCH --error=gpujob.%j.err # Stderr (%j expands to jobId)
#SBATCH --ntasks=4 # Total number of tasks
#SBATCH --gres=gpu:2 # GPUs per node
#SBATCH --nodes=4 # Total number of nodes requested
#SBATCH --ntasks-per-node=1 # Tasks per node
#SBATCH --cpus-per-task=20 # Threads per task
#SBATCH --mem=56000 # Memory per job in MB
#SBATCH -t 01:30:00 # Run time (hh:mm:ss)
#SBATCH --partition=gpu # Run on the GPU nodes queue
#SBATCH -A testproj # Accounting project


# Load any necessary modules


module load gnu/6
module load java/1.8.0
module load cuda/10.1.168
module load intel/18
```

```
module load intelmpi/2018

module load tensorflow/2.1.0


export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK


# Launch the executable

srun EXE ARGS
```

7.5.2 Pytorch

7.5.2.1 Introduction

PyTorch is a Python-based, open source machine learning framework, that is targeted at two sets of audiences:

- A replacement for NumPy to use the power of GPUs.
- A deep learning research platform that provides maximum flexibility and speed.

PyTorch has been primarily developed by Facebook's AI Research lab (FAIR).

More information about PyTorch can be found on the PyTorch webpage.

7.5.2.2 PyTorch Module

Usually on a supercomputer system multiple versions of PyTorch are available. Each of these versions requires different environment settings, for example different values of the $PATH and $LD_LIBRARY_PATH environment variables. The Environment Modules package is used to dynamically modify the user's environment according to the PyTorch (or other software) version that has to be used.

7.5.2.3 Use with Horovod

Horovod is included in the PRACE PyTorch modules. So it is possible to use Horovod after loading the PyTorch module, following the instructions in the PyTorch documentation.

7.5.2.4 Module Usage

To list all available modules on a system, use:

```
module avail
```

To list all available versions of PyTorch, use:

```
module avail pytorch
```

Example:

```
# module avail pytorch


-------------------- /apps/modulefiles/applications ----------------------

pytorch/1.1.0 pytorch/1.2.0 pytorch/1.3.1 pytorch/1.4.0 pytorch/1.5.0
```

To check module dependencies and the environment changes that will happen if a module is loaded, use '`module show`' . Example

```
# module show pytorch/1.5.0

-------------------------------------------------------------------

/apps/modulefiles/applications/pytorch/1.5.0:


module-whatis    Enable usage for Pytorch 1.5.0 with GNU Compilers 8 and CUDA 10.1

prereq   gnu/8

prereq   intel/18

prereq   intelmpi/2018

prereq   cuda/10.1.168

setenv           PYTORCHROOT /apps/applications/pytorch/1.5.0

prepend-path     PATH /apps/applications/pytorch/1.5.0/bin

prepend-path     LD_LIBRARY_PATH /apps/applications/pytorch/1.5.0/lib

prepend-path     LD_LIBRARY_PATH /apps/applications/pytorch/1.5.0/lib64

prepend-path     LD_LIBRARY_PATH /apps/applications/pytorch/1.5.0/lib/engines-1.1

prepend-path     LD_LIBRARY_PATH
/apps/applications/pytorch/1.5.0/lib/python3.7/lib-dynload

prepend-path     PYTHONPATH /apps/applications/pytorch/1.5.0/lib/python3.7/site-
packages

prepend-path     PYTHONPATH /apps/applications/pytorch/1.5.0/lib

prepend-path     PYTHONPATH /apps/applications/pytorch/1.5.0/lib64

prepend-path     PKG_CONFIG_PATH /apps/applications/pytorch/1.5.0/lib/pkgconfig

prepend-path     PKG_CONFIG_PATH /apps/applications/pytorch/1.5.0/lib64/pkgconfig

prepend-path     C_INCLUDE_PATH /apps/applications/pytorch/1.5.0/include

prepend-path     INCLUDE /apps/applications/pytorch/1.5.0/include

-------------------------------------------------------------------
```

To load a PyTorch module, run '`module load`', with the version you wish to load and its dependencies. For example, if we would like to load version 1.5.0 :

```
module load gnu/8 intel/18 intelmpi/2018 cuda/10.1.168 pytorch/1.5.0
```

To check loaded modules use '`module list`' :

```
# module list

Currently Loaded Modulefiles:

  1) gnu/8           2) intel/18        3) intelmpi/2018   4) cuda/10.1.168
5) pytorch/1.5.0
```

To clean your environment from all loaded modules use '`module purge`' :

```
# module purge

# module list
```

```
No Modulefiles Currently Loaded.
```

7.5.2.5 Job Submission

To run a job, a job script has to submitted to the system's Workload Manager, usually SLURM. An example of a SLURM batch script is provided below. However most details, like partition, account, tasks per node, will be different on each PRACE system. Please consult the system's documentation and support for the correct values.

```
#!/bin/bash -l

#-----------------------------------------------------------------
# GPU job on 4 nodes ,
# with 2 gpus per node and 20 threads per MPI task.
#-----------------------------------------------------------------

#SBATCH --job-name=gpujob # Job name
#SBATCH --output=gpujob.%j.out # Stdout (%j expands to jobId)
#SBATCH --error=gpujob.%j.err # Stderr (%j expands to jobId)
#SBATCH --ntasks=4 # Total number of tasks
#SBATCH --gres=gpu:2 # GPUs per node
#SBATCH --nodes=4 # Total number of nodes requested
#SBATCH --ntasks-per-node=1 # Tasks per node
#SBATCH --cpus-per-task=20 # Threads per task
#SBATCH --mem=56000 # Memory per job in MB
#SBATCH -t 01:30:00 # Run time (hh:mm:ss)
#SBATCH --partition=gpu # Run on the GPU nodes queue
#SBATCH -A testproj # Accounting project

# Load any necessary modules

module load gnu/8
module load intel/18
module load intelmpi/2018
module load cuda/10.1.168
module load pytorch/1.5.0

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

# Launch the executable
srun EXE ARGS
```

7.5.3 Spark

Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL and structured data processing, MLlib for Machine Learning, GraphX for graph processing, and Spark Streaming.

Figure 19 below, presents a visual representation of the components involved on how Spark runs on clusters. Spark applications run as independent sets of processes on a cluster, coordinated by the *SparkContext* object in the main program (called the *driver program*). The SparkContext can connect to several types of *cluster managers* (Spark's own standalone cluster manager, Mesos or YARN), which allocate resources across applications. Once connected, Spark acquires *executors* on nodes in the cluster, which are processes that run computations and store data for the application. Next, it sends your application code (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends *tasks* to the executors to run [94].

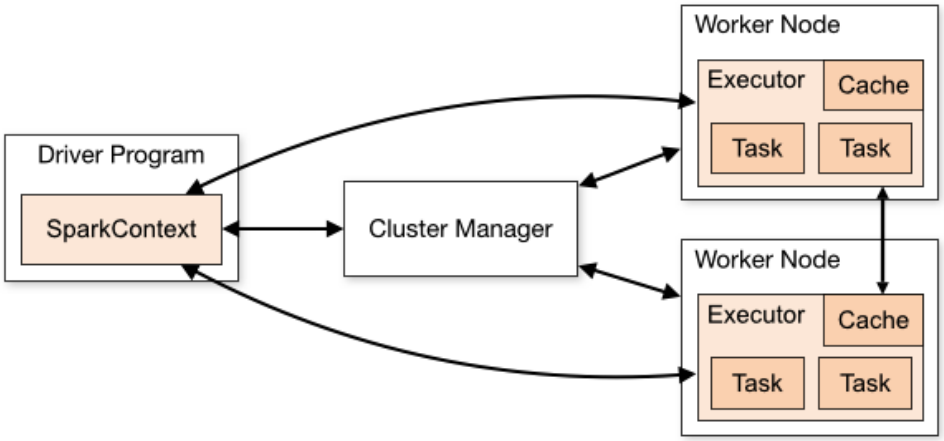

Figure 19 : The Spark architecture

This document describes all the steps necessary to create a multinode Spark standalone cluster (version 2.4) within a SLURM job. It has been tested on Cirrus [98], a Tier-2 machine hosted by EPCC. As with a lot of HPC services, Cirrus uses a Slurm scheduler to manage access to resources and schedule jobs. Writing a submission script is typically the most convenient way to submit your job to the scheduler.

7.5.3.1 Spark installation

Once in the download page, choose the Spark release and package type. For this document, we worked with Apache Spark 2.4.7 Pre-built for Apache Hadoop 2.6. Download Spark 2.4.7 and extract in your $HOME.

```
wget https://www.mirrorservice.org/sites/ftp.apache.org/spark/spark-
2.4.7/spark-2.4.7-bin-hadoop2.6.tgz

tar xvf spark-2.4.7-bin-hadoop2.6
```

You can start a standalone master server by executing:

```
./sbin/start-master.sh
```

Once started, the master will print out a *spark://HOST:PORT* URL for itself, which you can use to connect workers to it, or pass as the "master" argument to SparkContext. Similarly, you can start one or more workers and connect them to the master via:

```
./sbin/start-slave.sh <master-spark-URL>
```

For more information and arguments regarding starting a Cluster Manually check the Spark Standalone Mode site [95]. We have created the following bash scripts to be called from our slurm jobs later on.

Create a directory named bash_scripts in your $HOME and store the following bash files, which will be starting the master and worker nodes.

start_master.sh

```
#!/bin/bash
module load anaconda/python3
source activate cirrus-py36
export SPARK_HOME=${HOME}/spark-2.4.7-bin-hadoop2.6 cd $SPARK_HOME/
sbin/start-master.sh
echo "Started spark Master $HOSTNAME"
```

start_worker.sh

```
#!/bin/bash
hostmaster=$1
hostdriver=$2
export SPARK_HOME=${HOME}/spark-2.4.7-bin-hadoop2.6
export HOSTNAME=`hostname`

module load anaconda/python3
source activate cirrus-py36

echo "I am at " $HOSTNAME "and the master is " $hostmaster

if [ $HOSTNAME != $hostmaster ] && [ $HOSTNAME != $hostdriver ]

then

  echo "Started a WORKER on `hostname`"

  echo $HOSTNAME >> worker.log

cd $SPARK_HOME/
sbin/start-slave.sh $hostmaster:7077

else
  echo "Master" $hostmaster "or driver node" $hostdriver" - I dont start a
WORKER on " $HOSTNAME
```

```
fi
```

7.5.3.2 Create a conda environment

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. The following steps can be skipped, however, note that if you decide to follow our instructions, our scripts are activating and working within the following conda environment.

To create a python 3 environment in Cirrus, do:

```
module load anaconda/python3
conda create -n cirrus-py36 python=3.6 anaconda
```

To activate an active environment, use:

```
source activate cirrus-py3
```

To deactivate an active environment, use:

```
source deactivate
```

7.5.3.3 Start the Spark Cluster

To start the Spark cluster, submit a job submission script (in this case called: `sparkcluster_driver.slurm`) to the scheduler

```
sbatch sparkcluster_driver.slurm
```

Wait until the job is running before proceeding to run your Spark applications. You should modify the sparkcluster_driver.slurm according to your machine and need. For example, change the amount of time, number of nodes, account and/or the java path of your machine (unless you are using Cirrus). The current script on Cirrus configures a Spark cluster of 324 cores (9 nodes *X* 36 cores per node).

`sparkcluster_driver.slurm`

```
#!/bin/bash
#SBATCH --job-name=SPARKCLUSTER
#SBATCH --time=24:00:00
#SBATCH --exclusive
#SBATCH --nodes=9
#SBATCH --tasks-per-node=36
#SBATCH --cpus-per-task=1
#SBATCH --account=XXXX
#SBATCH --partition=standard
#SBATCH --qos=standard

module load spack
export JAVA_HOME=/lustre/sw/spack/opt/spack/linux-centos7-x86_64/gcc-
6.2.0/jdk-8u92-linux-x64-24xtmiygsdlaayomilfa5mnrasmxqlhj
module load anaconda/python3
source activate cirrus-py36
export SPARK_HOME=$HOME/spark-2.4.7-bin-hadoop2.6
export SPARK_MASTER_HOST=$HOSTNAME
```

```
export SPARK_MASTER_PORT=7077
export SPARK_MASTER_WEBUI_PORT=8080
export PATH=$SPARK_HOME/sbin:$SPARK_HOME/bin:$PATH

cd $HOME/bash_scripts rm -f master.log
rm -f driver.log
rm -f worker.log
rm -f nodes_list.log
echo "HOSTNAME is" $HOSTNAME
scontrol show hostnames $SLURM_JOB_NODELIST > nodes_list.log
mastername=$(head -n 1 nodes_list.log)
echo "master is " $mastername
echo $mastername > master.log
fileItemString=$(cat nodes_list.log |tr "\n" " ")
nodes=($fileItemString)
echo ${nodes[*]}
for each in "${nodes[@]}"
do

  echo "Nodo: $each" done

# start resource manager only once ./start master.sh
echo "Started the master" $mastername sleep 20s

drivername="NONE"
# start workers in all the nodes except the one where the master and driver
were started for i in "${nodes[@]}"
do

  echo $i

  ssh $i "cd $HOME/bash_scripts; ./start_worker.sh $mastername $drivername"
& done

#$SPARK_HOME/sbin/start-history-server.sh

sleep 24h
```

7.5.3.4 Launching Spark Applications

Spark comes with several sample programs. Scala, Java, Python and R examples are in the *examples/src/main* directory. To run one of the Java or Scala sample programs, use *bin/run-example <class> [params]* in the top-level Spark directory.

```
./bin/run-example SparkPi 10
```

For more details on submitting Spark applications check Launching Applications with spark-submit [96]. Attached a slurm job which will submit an application at the Spark Cluster created above. Replace the last line for the Spark application you want to launch.

```
#lauch_sparkapp.slurm
```

```
#!/bin/bash
#SBATCH --job-name=Round1 #SBATCH --time=00:05:00
#SBATCH --exclusive
#SBATCH --nodes=1
#SBATCH --tasks-per-node=36 #SBATCH --cpus-per-task=1
```

```
#SBATCH --account=XXXX #SBATCH --partition=standard
#SBATCH --qos=standard

module load spack
export JAVA_HOME=/lustre/sw/spack/opt/spack/linux-centos7-x86_64/gcc-
6.2.0/jdk-8u92-linux-x64-24xtmiygsdlaayomilfa5mnrasmxqlhj
module load anaconda/python3
source activate cirrus-py36

export SPARK HOME=$HOME/spark-2.4.7-bin-hadoop2.6
export SPARK_MASTER_HOST=$HOSTNAME
export SPARK_MASTER_PORT=7077
export SPARK_MASTER_WEBUI_PORT=8080
export PATH=$SPARK_HOME/sbin:$SPARK_HOME/bin:$PATH

hostmaster=$(cat "bash_scripts/master.log")
echo "Master Node" $hostmaster
export SPARK_HOME=${HOME}/spark-2.4.7-bin-hadoop2.6

NUM=$(wc -l $HOME/bash_scripts/worker.log)
NUMWORKERS=$(echo $NUM| cut -d' ' -f1)
NUMCORES=$( expr 36 '*' "$NUMWORKERS")

echo "Number of cores for this query is" $NUMCORES

### Launch your Spark Applications

$SPARK_HOME/bin/run-example SparkPi 10
```

This work used the Cirrus UK National Tier-2 HPC Service [97] at EPCC funded by the University of Edinburgh and EPSRC (EP/P020267/1)