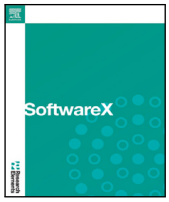
SoftwareX 24 (2023) 101538

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original Software Publication

PyOphidia: A Python library for High Performance Data Analytics at scale

Donatello Elia *, Cosimo Palazzo, Sandro Fiore[1], Alessandro D'Anca, Andrea Mariello[2], Giovanni Aloisio

Fondazione Centro Euro-Mediterraneo sui Cambiamenti Climatici, 73100, Lecce, Italy

ARTICLE INFO

Dataset link: https://github.com/OphidiaBigData/PyOphidia

Keywords:
High Performance Data Analytics
Scientific datacube
Climate analytics
Data science
Python module

ABSTRACT

The increasing size of scientific datasets has caused a deep transformation in the way scientific research is currently carried out. In multiple domains, Big Data challenges have called for novel software solutions capable of exploiting fast computing resources, workflows technologies and parallel paradigms at scale, by providing a high-level of abstraction while hiding, at the same time, the underlying infrastructural and software complexity from scientists. This paper describes PyOphidia, an open-source Python module for High Performance Data Analytics on multi-dimensional scientific datasets. PyOphidia aims to simplify the execution of parallel data analysis over scientific datacubes on High Performance Computing infrastructures. It can be easily integrated with other existing Python libraries and tools, within multiple data science environments.

Code metadata

Current code version	v1.11
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00522
Permanent link to Reproducible Capsule	
Legal Code License	GPLv3
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Python v3.7.0+, xarray v2022.03.0+ (optional), pandas 1.2+ (optional), numpy 1.19+ (optional), a running Ophidia server instance v1.8; any OS supporting Python
If available Link to developer documentation/manual	https://pyophidia.readthedocs.io/en/latest/
Support email for questions	ophidia-info@cmcc.it

1. Motivation and significance

Over the last two decades, the quantity of data available to scientists has grown at an unprecedented scale [1]. High-resolution simulations, observations from high-precision instruments and the huge amounts of data from sensors and edge devices have propelled this expansion [2,3]. The availability of such amounts of data opened the floor to the definition of more data-centric research paradigms besides more traditional compute-centric ones [4]. These changes had a disruptive impact on scientific research, which is now essentially relying on software technologies for taming the vast amount of data [5].

Together with new opportunities, Big Data brings forward multiple challenges that the scientific software community needs to address aiming to ensure efficient management throughout the entire data lifecycle [6–8]. For instance, in the Earth Sciences context, which is the main domain targeted by this work, the volumes of data available from model simulations and reanalysis are nowadays in the order of TeraBytes (i.e. output of a single simulation run), to PetaBytes (i.e. data published large community experiments or historical reanalysis datasets), heading towards ExaBytes in the near future (i.e. data produced in very high-resolution climate projections) [9–12]. Besides posing substantial technical challenges connected with the scale of such

Abbreviations: HPDA, High Performance Data Analytics; CMIP, Coupled Model Intercomparison Project

* Corresponding author.
E-mail address: donatello.elia@cmcc.it (Donatello Elia).
[1] Present Affiliation: University of Trento, 38122, Trento, Italy.
[2] Present Affiliation: ML Engineering Lead, Milan, Italy.

https://doi.org/10.1016/j.softx.2023.101538
Received 4 August 2023; Received in revised form 19 September 2023; Accepted 27 September 2023
Available online 10 October 2023

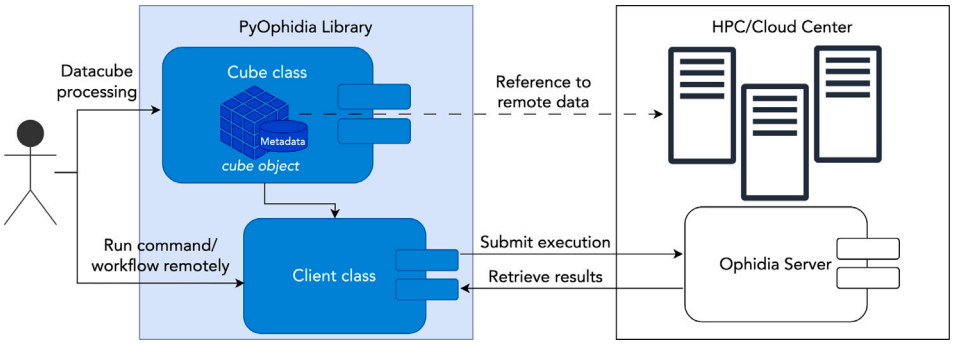


Fig. 1. Software architecture of the PyOphidia module, highlighting the interaction with the server-side infrastructure.

(big) data, climate datasets also need targeted software solutions able to deal with domain-specific aspects such as, among others, scientific data formats, legacy libraries and tools, controlled vocabularies and metadata management [13].

The aforementioned requirements have led to the development of the Ophidia Framework[3] in 2012, [14] and the PyOphidia Python module a few years later. Ophidia is an open-source framework aimed at providing High Performance Data Analytics (HPDA) capabilities for scientific applications involving large-scale multi-dimensional scientific data [15]. The framework has been primarily designed to address climate and geosciences needs, although it is flexible enough to support analysis of data from other scientific domains. The PyOphidia module brings HPDA capabilities within the scientific Python ecosystem through a high-level interface that allows handling distributed datasets, running parallel analysis and interacting with High Performance Computing (HPC) infrastructures.

The remainder of this paper is organized as follows: Section 2 presents the PyOphidia module, its architecture and capabilities, Section 3 showcases a relevant usage example, while Section 4 reports on the software impact and future work.

2. Software description

PyOphidia represents the Python bindings of the Ophidia framework, and it aims to provide a high-level and programmatic interface for data analytics at scale, while hiding the complexity of the underlying infrastructure from scientists. It is not just an interface towards the framework, but a powerful library for handling scientific data in the form of datacubes, as well as managing workflow execution, enabling parallel processing on HPC systems, and supporting integration with well-known modules from the Python scientific ecosystem.

PyOphidia is an open source software, released under GPLv3 license[4] and written entirely in Python, with the current version supporting up to Python v3.11. The source code is available on GitHub,[5] whereas the package can be retrieved from the Python Package Index (i.e., Pypi)[6] and the Conda Forge channel on Anaconda.[7] The module can be easily integrated with other scientific Python libraries such as those supported by the Pangeo project [16].

2.1. Software architecture

The overall PyOphidia software architecture is explained in Fig. 1. As it can be seen PyOphidia provides the client-side capabilities, usually running on the scientist's laptop, while the Ophidia Framework provides the server-side counterpart running on HPC or Cloud resources. The library consists of two main Python classes:

- *Client class*: it provides the basic functions to submit any command or workflow using a declarative query-like language offered by the Ophidia front-end server. This class exploits a custom implementation of a web service (according to WS-I+ standard [17]) based on gSOAP over HTTPS to perform the client–server interactions. By means of this client–server approach, the client class allows users to execute data analytics operators or shell commands on remote HPC and Cloud infrastructures. The connection is secured via SSL/TLS;
- *Cube class*: it implements the abstractions to handle scientific datasets (i.e., multi-dimensional data) as Python datacube objects and HPDA operators as Python datacube methods. The cube objects store the metadata and a reference towards the actual data managed by the server-side components. Moreover, the class implements a set of methods for materializing these virtual datacubes into (client-side) Python objects containing the whole data (according to the Ophidia data model [18]). The cube class also provides methods for translating these objects into common scientific Python formats, such as Xarray Datasets [19] or Pandas Dataframes [20]. The cube class interface has been designed to support HPDA tasks and deployment capabilities on top of HPC infrastructures. This class exploits the client class for the interaction with remote components, as shown in Fig. 1.

2.2. Data distribution and execution parallelism

The main goal of the PyOphidia library is to support HPDA on large scientific datasets. To this end, the library provides the capabilities for handling multi-dimensional data using the datacube abstraction borrowed from the datawarehouse context, jointly with a set of On-Line Analytical Processing (OLAP) operators. This abstraction is particularly suited for scientific data as these are inherently multi-dimensional. The Ophidia data model (widely discussed in [18]) implements the datacube abstraction and strategies to split and partition data horizontally into fragments across multiple computing nodes on the server-side. Towards enabling scalable analytics, the server-side components provide parallel access and processing to the distributed fragments composing a datacube.

From an implementation standpoint, Fig. 2 clarifies the key steps needed to run a server-side parallel data analysis on a HPC infrastructure. In particular, the sequence of steps is:

1. *Infrastructure deployment*: as a first step, PyOphidia implements the methods for firing up (i.e., deploying) the Ophidia computing components on the server-side (e.g., the nodes of a HPC cluster);
2. *Data access and loading*: PyOphidia provides methods to load data while specifying also the partitioning and distribution strategy; more specifically: (i) the number of fragments the datacube is split into and (ii) the number of nodes the fragments are distributed over;

[3] Ophidia: https://ophidia.cmcc.it/
[4] GNU General Public License v3: https://www.gnu.org/licenses/gpl-3.0.txt
[5] PyOphidia at: https://github.com/OphidiaBigData/PyOphidia
[6] PyOphidia on Pypi: https://pypi.org/project/PyOphidia/
[7] PyOphidia on Conda Forge: https://anaconda.org/conda-forge/pyophidia

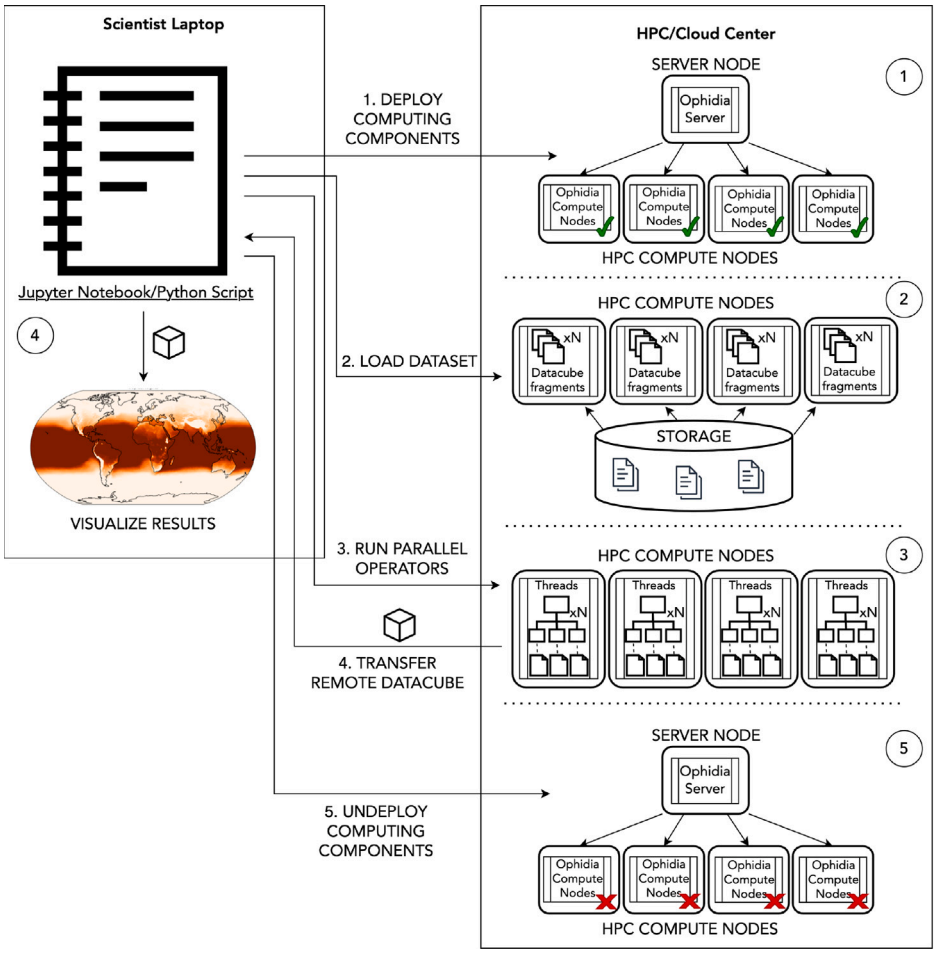


Fig. 2. Main steps in a HPDA application implemented with PyOphidia for running remote parallel processing on a HPC infrastructure [WITH COLORS].

3. *Parallel analytics*: the library provides analytics interfaces that allow defining the level of parallelism (i.e., number of processes and threads per processes) over the partitioned datacubes;
4. *Data transfer and visualization*: if needed, data can be transferred from the server-side running at the HPC center to the client side for further post-processing and visualization tasks, by using proper PyOphidia interfaces;
5. *Infrastructure un-deployment*: similarly to the first step, once the processing is completed, a method for shutting down (i.e, un-deploying) the Ophidia computing components on the server-side, is provided.

The Ophidia Framework provides multiple levels of parallelism: at *datacube-level*, i.e., multiple independent instances of the same operator can be run concurrently on different datacubes (or files to load); and at *fragment level*, i.e., a single operator is run in parallel exploiting a hybrid multi-process/multi-thread approach on the different datacube fragments. The interface of the parallel operators, and the related methods in PyOphidia, have been designed to provide a high degree of flexibility so that users or applications can tailor analytics execution according to the workload. Fig. 2 shows a simplified view of the server-side Ophidia Framework architecture; a detailed description can be found in [15].

2.3. Software functionalities

PyOphidia provides a wide set of functionalities for handling and processing large multi-dimensional datasets. The key methods implemented by the libraries support:

- connecting a remote Ophidia server instance and managing client–server interactions, including user authentication/authorization to the server side;
- submission of analytics query with the Ophidia query-like language, as well as complex workflows in JavaScript Object Notation (JSON) format (workflow format defined in [21]);
- handling virtual multi-dimensional data in the form of datacubes. To this end, the *cube* class supports the creation of datacube objects linked to the remote server-side data. The object includes a reference to the data via a unique identifier (called persistent identifier - PID), together with all the metadata associated with the remote datacube, such as the dimension, structure, size, length of fields, etc.
- operations on the data associated with the virtual cube objects. All these operators allow the user to specify the level of parallelism required to run operations remotely on the server-side infrastructure. The supported operations consist of:
  - data loading and storing from scientific binary formats such as NetCDF [22]. These methods provide the abstractions for defining the structure of remote datacubes in terms of partitioning and distribution on the server-side storage nodes;
  - statistical data reduction operations like average, max, min, standard deviation, etc.;
  - data subsetting on multiple dimensions;
  - predicate evaluation;
  - intercomparison and merging of multiple datacubes;
  - execution of user defined functions (called primitives in Ophidia);

- operations to handle datacube metadata and reorganize their internal structure. Moreover, the software supports community-based vocabularies like the Climate and Forecast Metadata Conventions [23];
- server-side interaction with remote HPC systems to (i) deploy and un-deploy the computing components of the Ophidia Framework, (ii) run the operations on the remote cube objects, and (iii) execute generic Python functions/scripts on the nodes;
- materialization of remote datacubes on the client-side into actual Python data in the PyOphidia multi-dimensional format. Moreover, the library provides methods for converting remote objects into client-side data in well-known scientific Python formats (i.e., Pandas DataFrames and Xarray Datasets).

3. Illustrative example

This section presents a typical example of how to use PyOphidia for the parallel computation of a climate indicator on a scientific dataset and exploiting HPC resources. Additional examples of Python codes and Jupyter Notebooks [24] based on PyOphidia can be found on the module documentation page[8].

Listing 1 shows how to compute the Tropical Nights extreme climate indicator [25] with PyOphidia, which is *defined as the annual count of days where the daily minimum temperature is above 20 °C (or 293.15 K)*. In particular, the first statement (line 2) establishes the connection towards the server front-end running on the remote HPC infrastructure. It is noteworthy that all PyOphidia methods, except from line 14, despite being executed locally, will be submitted to the front-end for remote execution.

```
1  from PyOphidia.cube import Cube
2  Cube.setclient(read_env=True)
3
4  Cube.cluster(action="deploy",host_partition="
       test_partition",nhost=4)
5
6  myCube1 = Cube.importnc(src_path="/tasmin_
       ESM2_ssp585_r1i1p1f1_gn_20900101-21001231.nc",
       measure="tasmin", imp_dim="time", description="Min
       Temperatures", host_partition="test_partition",
       nthreads=64)
7
8  myCube2 = myCube1.apply(query="oph_predicate(measure,'x
       -293.15','>0','1','0')", measure_type="auto",
       nthreads=64)
9
10 myCube3 = myCube2.reduce2(operation="sum", dim="time",
       concept_level="y", nthreads=64)
11
12 myCube4 = myCube3.subset(subset_dims="time",
       subset_filter=1)
13
14 pythonData = myCube4.to_dataset()
15
16 pythonData.tasmin.plot(cmap="Oranges", cbar_kwargs={"
       label": "Tropical Nights count"}, figsize=(16,8))
17
18 Cube.cluster(action="undeploy",host_partition="
       test_partition")
```

Listing 1: Example of Python code exploiting the PyOphidia module for running the computation of a climate index in parallel over a large dataset on a HPC infrastructure

Before running the actual analysis, the framework components are deployed on 4 nodes of the HPC infrastructure (line 4); a similar method can be used to un-deploy the components and release the resources (line 18). These methods allow controlling the amount of resources needed for the execution of the analysis without having to directly log into the remote HPC infrastructure, as described in Section 2.2.

Analytics operators can be then specified with the related method. Each method allows the user to specify the overall level of parallelism for the operator execution: in the case of this example, 64 threads are used in total (defined through the *nthreads* argument), so that 16 parallel threads are running on each of the 4 nodes handling a portion of the datacube fragments (i.e., using the fragment-level parallelism explained in Section 2.2).

The data loading operator (i.e., *importnc* method) allows defining data partitioning and distribution directly while data is read from the file system: users can define the number of nodes that data is distributed on (*nhost* parameter) and how the data is partitioned, by defining the number of data fragments per node (*nfrags* parameter). It is worth noting that users do not need to specify data partitioning arguments as the framework implements heuristics to automatically infer fragmentation according to the available resources and input data structure. In this specific example, a CMCC dataset from the CMIP6 repository [26] containing the minimum daily temperatures is imported in line 6: the resulting datacube is partitioned on the 4 nodes where the Ophidia components were previously deployed (i.e., identified by the *test_partition* name). Since the number of hosts and the number of fragments are not defined, the heuristics partition the datacube across all the nodes associated with the *test_partition*.

After the data is loaded on the HPC nodes, three parallel operators are applied to (i) extract the number of days over the given threshold 293.15 K (line 8), (ii) count the days for each year in the datacube (line 10), and extract the results for the first year (line 12).

Finally, using the *to_dataset* method (line 14) the datacube content can be transferred from the server-side infrastructure to the client-side and converted into an Xarray Dataset. It should be noted that at this stage the data being moved from the remote server to the notebook is quite small (less than a MegaByte in this example). The results from this command can be accessed directly within the Python script, locally, for further processing and visualization (line 16). Fig. 3 shows a plot with the result from the Tropical Night indicator produced with the Xarray built-in plotting functions based on Matplotlib [27]. As it can be argued, the methods for converting Ophidia datacubes into other formats enable a seamless integration of PyOphidia HPDA features with other scientific Python libraries.

4. Impact and conclusions

The PyOphidia module aims to address the challenges related to the increasing volumes of scientific data by providing a high-level solution tailored for parallel processing of multi-dimensional data. The module provides the abstractions for handling data parallelism on heterogeneous computing environments (both HPC and Cloud), trying to hide the complexity of the underlying infrastructure, as shown in Section 3. Extensive documentation, along with examples of usage and video tutorials/materials, and a Docker container (hosting both PyOphidia and the Ophidia software stack) targeting in particular the Earth Science domain, have been made available online to help scientists take advantage of the package[9]. Moreover, PyOphidia can be used in conjunction with other Python modules and tools for interactive analysis and visualization.

PyOphidia was released to the public in 2016. It has been improved and supported by multiple developers ever since, and exploited by several users throughout the years. The current version (v1.11) comprises

[8] PyOhidia docs: https://pyophidia.readthedocs.io/en/latest/

[9] PyOphidia tutorials: https://pyophidia.readthedocs.io/en/latest/tutorial.html

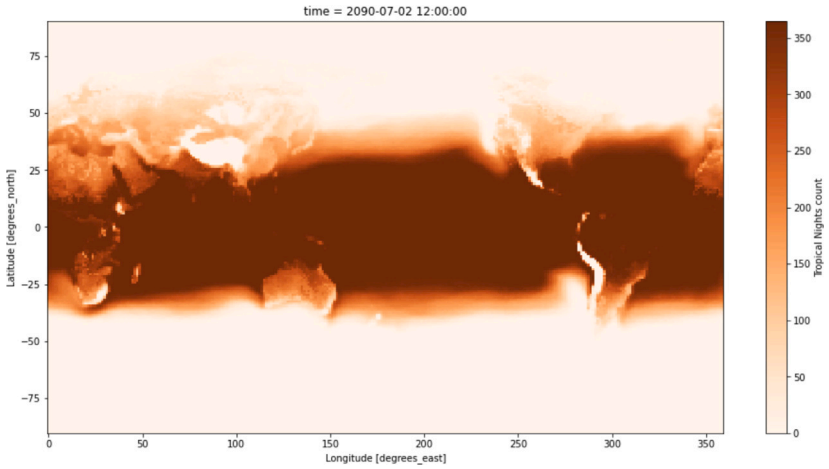


Fig. 3. Color mesh plot showing the results of the Tropical Nights indicator computation. Figure produced using the built-in plotting functions from Xarray/Matplotlib [WITH COLORS].

more than 5300 lines of codes with over 230 commits on the GitHub repository.

The Python module has been used, jointly with the Ophidia server-side components, in a wide set of applications, mainly related to climate and geosciences [28–30] but also linked with biodiversity and smart cities [31,32]. Other than being used by scientists on a daily basis, the module has also been exploited in several applications developed in the context of international and European projects. In particular, the module is currently used in the European projects eFlows4HPC [33], interTwin[10] and ESiWACE Center of Excellence,[11] and it is one of the key components of the Data Space science gateway for the European Network for Earth System Modelling (ENES) community [34].

Overall, the PyOphidia module enables greater productivity of scientists, as they can focus more on the science part rather than the setup of the system and the interaction with the infrastructure, and it supports processing of parallel analytics for scientific datasets. PyOphidia capabilities have therefore proven useful for effective knowledge extraction on Big Data and, in turn, to support scientific discovery. Concerning future work, the module will undergo continuous developments to account for new requirements from the scientific community, support direct integration with other well-known community libraries, and increase the capabilities to handle complex and large workflows including thousands of tasks.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The code is available at: https://github.com/OphidiaBigData/PyOphidia

Acknowledgments

The development of PyOphidia has been in part supported by multiple research projects through the years; the full list of projects providing financial support is provided at: https://github.com/OphidiaBigData/PyOphidia/blob/master/NOTICE.rst. The most recent projects that provided support to PyOphidia are eFlows4HPC and ESiWACE2. eFlows4HPC receives funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No. 955558. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, and Norway. In Italy, it has been preliminarily approved for complimentary funding by Ministero dello Sviluppo Economico (MiSE) (ref. project prop. 2659). The ESiWACE2 project received funding under the European Union's Horizon 2020 Research and Innovation Programme under Grant No. 823988. The publication of this article was funded by the eFlows4HPC project.

The authors would like to thank all the people who contributed to the development of the module throughout the years. The full list of contributors is provided at https://github.com/OphidiaBigData/PyOphidia/blob/master/AUTHORS.rst.

[10] interTwin: https://www.intertwin.eu/

[11] ESiWACE: https://www.esiwace.eu/

References

[1] Philip Chen C, Zhang C-Y. Data-intensive applications, challenges, techniques and technologies: A survey on big data. Inform Sci 2014;275:314–47. http://dx.doi.org/10.1016/j.ins.2014.01.015.

[2] Overpeck JT, Meehl GA, Bony S, Easterling DR. Climate data challenges in the 21st century. Science 2011;331(6018):700–2. http://dx.doi.org/10.1126/science.1197869.

[3] Jagadish HV, Gehrke J, Labrinidis A, Papakonstantinou Y, Patel JM, Ramakrishnan R, et al. Big data and its technical challenges. Commun ACM 2014;57(7):86–94. http://dx.doi.org/10.1145/2611567.

[4] Bell G, Hey T, Szalay A. Beyond the data deluge. Science 2009;323(5919):1297–8. http://dx.doi.org/10.1126/science.1170411.

[5] Gray J, Liu DT, Nieto-Santisteban M, Szalay A, DeWitt DJ, Heber G. Scientific data management in the coming decade. SIGMOD Rec 2005;34(4):34–41. http://dx.doi.org/10.1145/1107499.1107503.

[6] Marx V. The big challenges of big data. Nature 2013;498(7453):255–60. http://dx.doi.org/10.1038/498255a.

[7] Hu H, Wen Y, Chua T, Li X. Toward scalable systems for big data analytics: A technology tutorial. IEEE Access 2014;2:652–87. http://dx.doi.org/10.1109/ACCESS.2014.2332453.

[8] Bethel EW, Greenwald M, van Dam KK, Parashar M, Wild SM, Wiley HS. Management, analysis, and visualization of experimental and observational data - the convergence of data and computing. In: 2016 IEEE 12th international conference on e-Science. 2016, p. 213–22. http://dx.doi.org/10.1109/eScience.2016.7870902.

[9] Deser C, Lehner F, Rodgers K, Ault T, Delworth T, DiNezio P, et al. Insights from earth system model initial-condition large ensembles and future prospects. Nature Clim Change 2020;10(4):277–86. http://dx.doi.org/10.1038/s41558-020-0731-2.

[10] Balaji V, Taylor KE, Juckes M, Lawrence BN, Durack PJ, Lautenschlager M, et al. Requirements for a global data infrastructure in support of CMIP6. Geosci Model Dev 2018;11(9):3659–80. http://dx.doi.org/10.5194/gmd-11-3659-2018.

[11] Hersbach H, Bell B, Berrisford P, Hirahara S, Horányi A, Muñoz-Sabater J, et al. The era5 global reanalysis. Q J R Meteorol Soc 2020;146(730):1999–2049. http://dx.doi.org/10.1002/qj.3803.

[12] Schär C, Fuhrer O, Arteaga A, Ban N, Charpilloz C, Girolamo SD, et al. Kilometer-scale climate models: Prospects and challenges. Bull Am Meteorol Soc 2020;101(5):E567–87. http://dx.doi.org/10.1175/BAMS-D-18-0167.1.

[13] Ailamaki A, Kantere V, Dash D. Managing scientific data. Commun ACM 2010;53(6):68–78. http://dx.doi.org/10.1145/1743546.1743568.

[14] Fiore S, D'Anca A, Palazzo C, Foster I, Williams D, Aloisio G. Ophidia: Toward big data analytics for escience. Procedia Comput Sci 2013;18:2376–85. http://dx.doi.org/10.1016/j.procs.2013.05.409, 2013 International Conference on Computational Science.

[15] Elia D, Fiore S, Aloisio G. Towards HPC and big data analytics convergence: Design and experimental evaluation of a hpda framework for escience at scale. IEEE Access 2021;9:73307–26. http://dx.doi.org/10.1109/ACCESS.2021.3079139.

[16] Hamman J, Rocklin M, Abernathy R. Pangeo: A big-data ecosystem for scalable earth system science. In: EGU General Assembly conference abstracts. 2018, p. 12146, URL https://ui.adsabs.harvard.edu/abs/2018EGUGA.2012146H.

[17] Chumbley R, Durand J, Hainline M, Pilz G, Rutt T. Basic profile version 1.2. Web Service Interoperability Organization; 2010, URL http://ws-i.org/Profiles/BasicProfile-1.2-2010-11-09.html.

[18] Fiore S, Elia D, Palazzo C, Antonio F, D'Anca A, Foster I, et al. Towards high performance data analytics for climate change. In: Weiland M, Juckeland G, Alam S, Jagode H, editors. High performance computing. Cham, Switzerland: Springer Nature Switzerland AG; 2019, p. 240–57. http://dx.doi.org/10.1007/978-3-030-34356-9_20.

[19] Hoyer S, Hamman J. xarray: N-D labeled arrays and datasets in Python. J Open Res Softw 2017;5(1). http://dx.doi.org/10.5334/jors.148.

[20] McKinney Wes. Data structures for statistical computing in python. In: van der Walt Stéfan, Millman Jarrod, editors. Proceedings of the 9th Python in science conference. 2010, p. 56–61. http://dx.doi.org/10.25080/Majora-92bf1922-00a.

[21] Palazzo C, Mariello A, Fiore S, D'Anca A, Elia D, Williams DN, et al. A workflow-enabled big data analytics software stack for escience. In: 2015 international conference on high performance computing & simulation. 2015, p. 545–52. http://dx.doi.org/10.1109/HPCSim.2015.7237088.

[22] Rew R, Davis G. NetCDF: an interface for scientific data access. IEEE Comput Graph Appl 1990;10(4):76–82. http://dx.doi.org/10.1109/38.56302.

[23] Gregory J. The CF metadata standard. In: CLIVAR exchanges, vol. 8, no. 4. 2003, p. 4.

[24] Kluyver T, Ragan-Kelley B, Pérez F, Granger B, Bussonnier M, Frederic J, et al. Jupyter notebooks – a publishing format for reproducible computational workflows. In: F. Loizides B Schmidt, editor. Positioning and power in academic publishing: Players, agents and agendas. IOS Press; 2016, p. 87–90. http://dx.doi.org/10.3233/978-1-61499-649-1-87.

[25] Klein Tank A, Zwiers F, Zhang X. Guidelines on analysis of extremes in a changing climate in support of informed decisions for adaptation. Tech. rep. WCDMP No. 72. WMO-TD No. 1500, World Meteorological Organization; 2009.

[26] Scoccimarro E, Bellucci A, Peano D. Cmcc cmcc-cm2-vhr4 model output prepared for cmip6 highresmip. 2017, http://dx.doi.org/10.22033/ESGF/CMIP6.1367.

[27] Hunter JD. Matplotlib: A 2D graphics environment. Comput Sci Eng 2007;9(3):90–5. http://dx.doi.org/10.1109/MCSE.2007.55.

[28] Fiore S, Płóciennik M, Doutriaux C, Palazzo C, Boutte J, Zok T, et al. Distributed and cloud-based multi-model analytics experiments on large volumes of climate change data in the earth system grid federation eco-system. In: 2016 IEEE international conference on big data. 2016, p. 2911–8. http://dx.doi.org/10.1109/BigData.2016.7840941.

[29] D'Anca A, Palazzo C, Elia D, Fiore S, Bistinas I, Böttcher K, et al. On the use of in-memory analytics workflows to compute eScience indicators from large climate datasets. In: 2017 17th IEEE/ACM international symposium on cluster, cloud and grid computing. 2017, p. 1035–43. http://dx.doi.org/10.1109/CCGRID.2017.132.

[30] Fiore S, Elia D, Palazzo C, D'Anca A, Antonio F, Williams DN, et al. Towards an open (data) science analytics-hub for reproducible multi-model climate analysis at scale. In: 2018 IEEE international conference on big data. 2018, p. 3226–34. http://dx.doi.org/10.1109/BigData.2018.8622205.

[31] Fiore S, Palazzo C, D'Anca A, Elia D, Londero E, Knapic C, et al. Big data analytics on large-scale scientific datasets in the INDIGO-DataCloud project. In: Proceedings of the computing frontiers conference. 2017, p. 343–8. http://dx.doi.org/10.1145/3075564.3078884.

[32] Fiore S, Elia D, Pires CE, Mestre DG, Cappiello C, Vitali M, et al. An integrated big and fast data analytics platform for smart urban transportation management. IEEE Access 2019;7:117652–77. http://dx.doi.org/10.1109/ACCESS.2019.2936941.

[33] Ejarque J, Badia RM, Albertin L, Aloisio G, Baglione E, Becerra Y, et al. Enabling dynamic and intelligent workflows for hpc, data analytics, and ai convergence. Future Gener Comput Syst 2022;134:414–29. http://dx.doi.org/10.1016/j.future.2022.04.014.

[34] Elia D, Antonio F, Fiore S, Nassisi P, Aloisio G. A data space for climate science in the European open science cloud. Comput Sci Eng 2023;(01):1–10. http://dx.doi.org/10.1109/MCSE.2023.3274047.