# Writing Efficient Computational Workflows in Python

Eduard Ayguadé, Rosa M. Badia, Yolanda Becerra, David Carrera, Toni Cortés, Jesús Labarta, Anna Queralt, Enric Tejedor, Jordi Torres, and Mateo Valero

Barcelona Supercomputing Center (BSC-CNS)

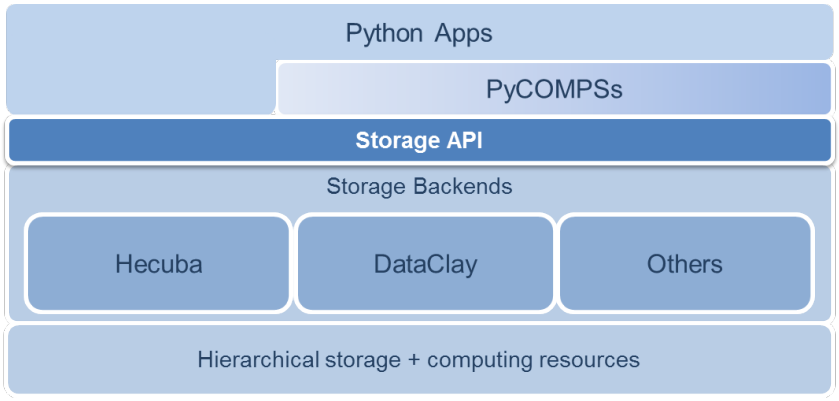
BSC presented its point of view on big data and exascale computing in the last BDEC meeting [1]. In this document, five different strategic considerations were outlined:

1. The importance of algorithmic optimization and tuning
2. The increasingly dynamic usage patterns of the infrastructures
3. The importance of providing clean programming interfaces integrating the concurrency and data management models
4. The responsibility of the runtime in optimizing the mapping of computation and data to available resources
5. The need to develop architectural support for the runtime and application functionalities

This white paper can be seen as a continuation of [1] extending the strategic consideration 3 that identified the importance of proposing programming models that **better integrate the concurrency and data processing** aspects trying to rely on the same type of abstractions at the different granularity levels. Particularly relevant is the integration of flexible parallel control flow structures and query mechanisms to refer to huge data sets. Support for flexible parallelization strategies (asynchrony, nesting) is needed to free applications from latency limitations, converting them into throughput-based applications where amount of resources (bandwidth, cores, $) is the limiting factor. By using **a new generation of object-based storage** layers we have the potential of closely mapping the data models in the programming models to those used to provide data durability and also to provide a flexible shared communication space between partially coupled or independent applications. Such integration would simplify programs, eliminating the programmer need to consider the different models and a lot of the code that today is devoted to explicit I/O operations.

A strategic initiative at BSC is the Severo Ochoa project, which fosters the integration and cooperation of the different departments at BSC to develop strategic applications and technologies. The project involves applications from the Life Science, Earth Science and Computer Applications department and technologies and tools from the Computer Science department: programming models, storage systems and resource management. In this context, a close loop is followed were applications define the requirements of the software stack, developments in the software stack components are performed following these requirements and next verified in the applications, which can define new requirements in the next cycle.

Our global objective is to devise a novel object storage layer interface that can be mapped on top of different underlying data management infrastructures and cleanly integrated with the programming model developments described in the following paragraphs.




The use of the Python programming language for scientific computing has been gaining momentum in the last years. The fact that it is compact and readable and its complete set of scientific libraries are two important characteristics that favor its adoption. Indeed, it is not only its large adoption by different scientific communities, but also its adoption by HPC-related scientific communities. For example, it is relevant that in the last SC14 event, one of its tutorials and one of its BoF was directly related to the use of Python in HPC.

At the programming model level, the **StarSs concept** is that the programmer specifies tasks and the directionality of the data accesses they perform. COMPSs [2] is the instance of the StarSs programming model focusing medium/coarse grain level. The model allows for very dynamic mapping of computations and data accesses to the available resources by an intelligent runtime and data management layers that are provided.

COMPSs [3] aim is to support flexible computational workflows. Several language bindings are available: C, C++ and Java for the original implementation and PyCOMPSs, introduces efficient parallel support in Python. The annotations of argument directionalities are provided through decorators in Python. In the current version, the arguments to the tasks can be files and objects declared in the language type model and dependences are also computed based on accesses to such local files and objects. The execution engine actually offloads the data objects and computations to different cores or nodes, supporting the efficient parallel execution of medium granularity programs. Tasks from the same computational workflow can actually be offloaded to different nodes within the local cluster or to external cloud resources in a transparent way.

Support in PyCOMPSs to Big Data has been incorporated through a Storage API, which has been defined to integrate the programming model with a persistent storage object model, abstracting it from the backend storage solution. The storage API enables Python scripts to create, delete, insert, retrieve and iterate over persistent data. At its turn, PyCOMPSs also invokes the Storage API, mainly to obtain locality information about persistent data.

The benefits of this integration goes beyond the enablement to access persistent data, which enables several applications to share data in a concurrent way, but to the enablement to access a larger amount of data than the one addressable in a single node memory: with PyCOMPSs a sequential Python script is run in parallel in a set of nodes of a cluster and it is able to transparently address data stored in large databases.

Currently two storage backends developed at BSC are available: Hecuba and dataClay. Hecuba is a module that aims to facilitate programmers an efficient and easy interaction with non-relational databases. For example, Hecuba redefines Python iterators, enabling regular Python code to access the persistent storage system. dataClay is a platform that enables application developers to manage persistent data following the Object Oriented (OO) paradigm. With dataClay, users can define sharing policies to their data models. Also, enrichment of existing classes is enabled in dataClay: users can add new properties, methods or implementations to existing methods.

PyCOMPSs offers an elegant solution to automatically parallelize/distribute applications in the widely used Python language. By cleanly integrating the persistent object model in it we expect to offer to application programmers the possibility to incrementally enable Big Data requirements in existing applications. The environment will also ease and encourage the productive programming of more dynamic workflows.

[1] Jesus Labarta, Eduard Ayguade, Fabrizio Gagliardi, Rosa M. Badia, Toni Cortes, Jordi Torres, Adrian Cristal, Osman Unsal, David Carrera, Yolanda Becerra, Enric Tejedor , Mateo Valero, "BSC vision on Big Data and extreme scale computing", BDEC Kukuoka white paper, 2014.

[2] Lordan F, Tejedor E, Ejarque J, Rafanell R, Álvarez J, Marozzo F, Lezzi D, Sirvent R, Talia D, Badia RM. ServiceSs: an interoperable programming framework for the Cloud. Journal of Grid Computing, 2014 ;12:67-91. Available from: http://hpc.ac.upc.edu/PDFs/dir21/file004255.pdf

[3] www.bsc.es/compss