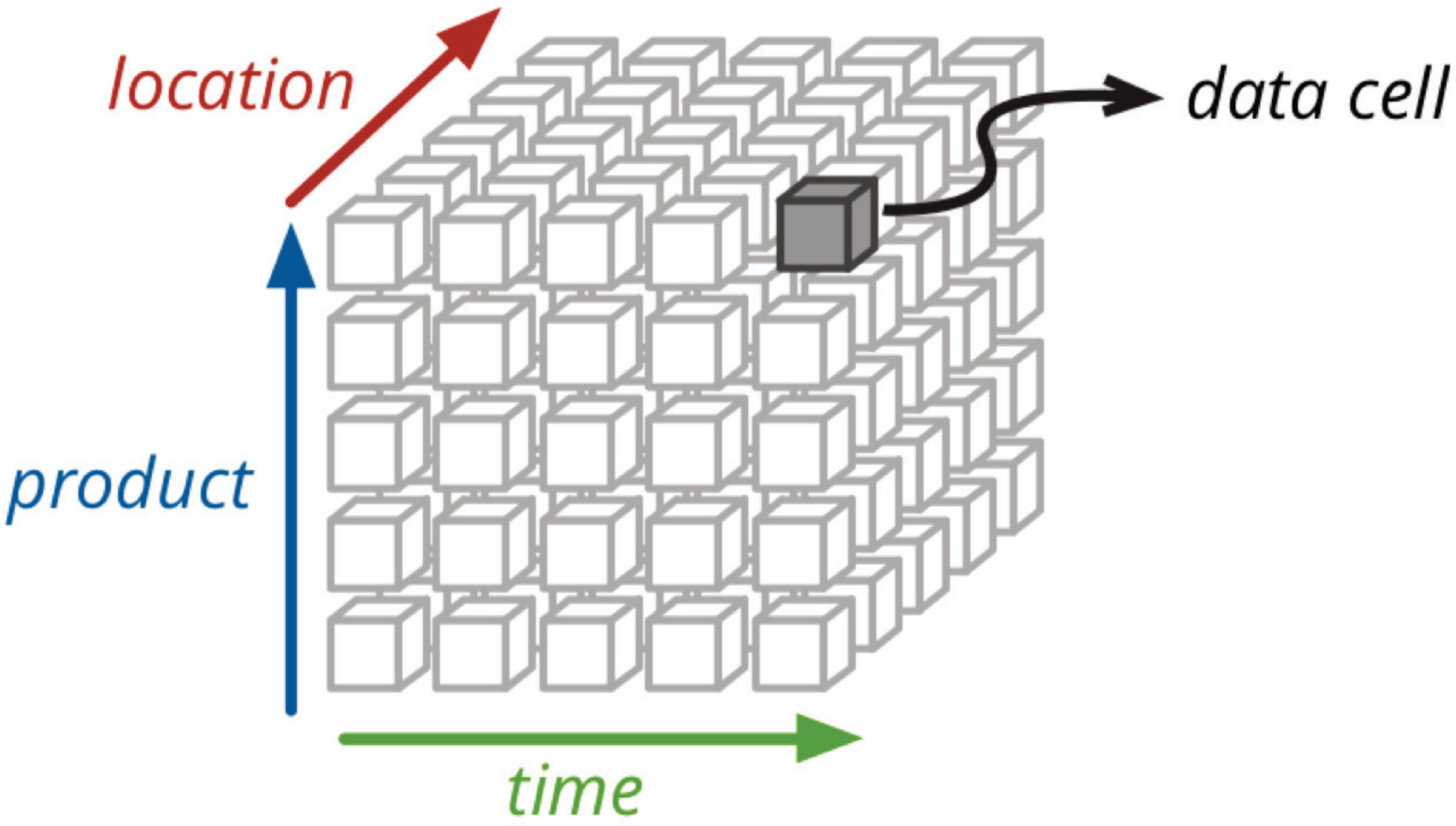
ISSN 2039-7941
anno 2019 numero 410



RAPPORTI
TECNICI INGV

A procedure to import seismological data into the Ophidia big data analytics framework

410

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

REGISTRAZIONE AL TRIBUNALE DI ROMA N.174 | 2014, 23 LUGLIO

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

ISSN 2039-7941
anno **2019** numero **410**

RAPPORTI
TECNICI INGV

A procedure to import seismological data into the Ophidia big data analytics framework

Nicola Mario Marcucci[1], Alessandro D'Anca[2], Stephen Monna[1], Sandro Fiore[2]

[1]INGV | Istituto Nazionale di Geofisica e Vulcanologia, Sezione Geomagnetismo, Aeronomia e Geofisica Ambientale
[2]CMCC | Fondazione Centro Euro-Mediterraneo sui Cambiamenti Climatici

Accepted 1st July 2019 | *Accettato 1 luglio 2019*
How to cite | *Come citare* Marcucci N. M., D'Anca A., Monna S., Fiore S., (2019). A procedure to import seismological data into the Ophidia big data analytics framework. Rapp. Tec. INGV, 410: 1-22.

Cover Datacube example | *In copertina Datacube example*

410

INDEX

Abstract

The term Big Data identifies large data sets that cannot be managed with traditional processing software. Volume, Variety and Velocity are properties used to classify Big Data. Velocity, the most relevant property, refers to high speed of data acquisition and processing. Management of Big Data, and particularly Velocity, has become an open issue for scientific communities. The EC project INDIGO-Datacloud was conceived to tackle some of the Big Data challenges that scientific communities are facing. In the context of INDIGO-Datacloud we defined a case study in which one of the goals was the improvement of processing performance on seismological data. Starting with a practical case study, we identified, as possible software solution, Ophidia, a big data analytics framework. The first step to exploit the framework functionalities required the import of the seismological data in Ophidia. For this purpose, we designed and implemented a software module that enables the import of seismological data in SAC format. As a second step, we tested the capabilities of Ophidia through a performance test to evaluate the framework potential.

Introduction

The European Commission is focusing its strategic efforts on the challenges of the Big Data era. The INDIGO-DataCloud project (INtegrating Distributed data Infrastructures for Global ExplOitation, INDIGO for short), funded by the Horizon 2020 program, is one such of these efforts. The mission of INDIGO was to develop a data and computing platform aimed at scientific communities. The main objective of INDIGO was to identify and meet the requirements of different scientific communities. Each scientific community has provided feedback and participated in the review of the services. Through the definition of a case study, communities selected, tested and implemented the solutions that best meet their needs. The National Institute of Geophysics and Volcanology (INGV), one of the communities involved (environmental sector), participated by proposing a case study based on seismological data recorded by multi-parameter seafloor observatories. One of the needs of the INGV working group, illustrated in our case study, was the improvement of computing performance, and the solution identified in INDIGO was the Ophidia framework.
The Ophidia framework provides a big data analytics platform addressing scientific use cases that include large volumes of multidimensional data. Ophidia has been created as a tool for analyzing climatological data in the Network Common Data Form (NetCDF) format. The import operator for seismological data, in particular the Seismic Analysis Code (SAC) format, was not available.
In this paper, we start with a general description of the Ophidia framework and its software infrastructure, illustrating the software stack from level 0 (I / O nodes layer) to level 4 (Front-End). The most important operators and primitives, provided by the framework, will be presented. We will explain the steps that have led to the implementation of the OPH_IMPORTSAC operator. Through SAC data import, we are now able to exploit Ophidia's computational parallelism to analyze seismological data and implement its algorithms.
The last section of the paper provides a performance test comparing SAC and Ophidia in the calculation of the Fast Fourier Transform.

1. INDIGO-DataCloud project

INDIGO's main objective was to develop a computing and data management platform that could be distributed across multiple hardware and provided on hybrid e-infrastructures (private or

public) [INDIGO, 2017]. INDIGO has exploited the formidable know-how acquired in Europe in the last 10 years by several groups that had dealt with scientific computing based on consolidated and emerging paradigms such as HPC, Grid and Cloud. In cloud computing, both the public and private sectors are already able to offer Infrastructure as a Service (IaaS) cloud resources. However, there are many areas of interest for scientific communities where cloud computing is currently lacking, especially at Platform as a Service (PaaS) and Software as a Service (SaaS) level. For this reason, INDIGO plan was to develop tools and platforms based on open source solutions, addressing scientific challenges in the areas of cloud computing, storage and networks. INDIGO has enabled the development and execution of applications based on cloud and grid infrastructures, as well as on HPC clusters. The project extended existing PaaS solutions, allowing public and private e-Infrastructures to integrate their existing services, making them available through federated and distributed AA policies, ensuring transparency and trust in the provision of such services.
INDIGO has also created a flexible and modular presentation layer, linked to the PaaS and SaaS frameworks developed within the project, which allows for an innovative user experience also from mobile devices.

2. Case study

Our Case Study (CS), within INDIGO, includes research activities performed at INGV on data recorded by seafloor observatories. The observatories are EMSO (European Multidisciplinary Seafloor and water column Observatory) nodes. EMSO is composed of several deep-seafloor and water column observatories, deployed at key sites in the European waters, thus forming a widely distributed pan-European infrastructure [EMSO, 2017].
The processed data in the CS were collected by the NEMO-SN1 observatory, located in the Western Ionian Sea, in proximity to Mount Etna volcano, at 2100 m below sea level (bsl). The raw data is in the form of long time series (> = 1 year). Some instruments on board the observatory produce large datasets, for example bio-acoustic hydrophones sampling at tens of kHz.
Researchers apply a quality control on raw data to identify possible spikes, holes, gaps and other anomalies that might be present.
Data analysis requires several steps that use a number of software codes and tools. In this chain of events, some operations may require significant computing power, although, the software is often run on the PCs of individual researchers. Furthermore, researchers use MOIST (Oceanic Multidisciplinary Information System) [MOIST, 2016] to store and visualize data and metadata. MOIST does not offer advanced data analysis tools.
On the basis of our CS, we have identified the requirements that were subsequently provided to the INDIGO developers and tested the proposed solutions that are of interest to our research community [Fiore et al., 2017].
At this stage, we focused on seismological data and the development of an operator to import Seismic Analysis Code (SAC) data in the Ophidia framework [OPHIDIA, 2017]. The name SAC identifies both the software and data format used by our group in seismological processing.

3. Seismic Analysis Code (SAC)

SAC was developed at the Lawrence Livermore National Laboratory (the copyright is owned by the University of California), and it is maintained by a small group of developers in collaboration with IRIS (https://www.iris.edu/hq/).
This section is based on information extracted from the IRIS website [IRIS, 2017].

3.1 SAC software

SAC "is an interactive program designed for the study of sequential signals, in particular time series data. The emphasis has been placed on the analysis tools used by seismologists in the detailed study of seismic events. The analyses that can be performed with SAC include general arithmetic operations, Fourier transformations, spectral estimation techniques, IIR and FIR filtering, stacking, decimation, interpolation, correlation and seismic phase identification" [IRIS, 2017]. SAC also contains extensive graphic capabilities. It is one of the most used seismology analysis tools. The version used in our CS was 101.6a.

3.2 SAC data format

This section describes the contents of the SAC data file in binary format and documents its header (the dataless part). "Each signal is stored on the disk in a separate SAC data file. These files contain a fixed-length header section followed by one or two data sections. The header contains floating-point, integer, logical and character fields. Evenly spaced data files have only one data section which contains the dependent variable. Unevenly spaced data and spectral data files contain two data sections. In this work, we used evenly spaced data files in binary format" (constant sampling interval of 0.01 s) [IRIS-2, 2017].
"The files in binary format can be read quickly from the disk in memory. The header is 158 words at 32 bits in length and is followed by the data section. To quickly read only a small section of a data file, these files also have a record length of 512 bytes (128 words at 32 bits) and are opened for direct access. There is no physical record structure" [IRIS-2, 2017]. This format is shown schematically in the schema 1 below.

Header Section	First Data Section
start word: 0	start word: 158
Word length: 158	Word length: NPTS

Scheme 1 SAC binary format.

The following is the list of header variables used in this work. For more details, refer to the SAC manual [IRIS-2, 2017].
Float variables:

- DELTA: dataset sampling rate;
- F: used to introduce an artificial variable, which we will describe later;

Integer variables:

- NZYEAR year in which the dataset starts;
- NZJDAY day in which the dataset starts, expressed in the Julian calendar;
- NZHOUR start time;
- NZMIN start minute;
- NZSEC start second;
- NZMSEC start millisecond;
- NPTS: number of samples.

The metadata listed above are used in the import phase.

4. Ophidia Big Data Analytics Framework

The **Ophidia** project provides a **big data analytics** platform addressing scientific use cases that include large volumes of multidimensional data. This section provides a general description of the Ophidia framework and its software infrastructure, from level 0 (I / O nodes layer) to level 4 (Front-End). The most important operators and primitives provided for by the framework are introduced.

4.1 Ophidia

Ophidia is a big data analytics framework for eScience that provides a stacked software infrastructure to store and manipulate scientific data in datacube form [Fiore et al., 2013]. A datacube is a multidimensional array of values, commonly used to describe a series of (usually temporal) data.
In the scientific big data scenario, the Ophidia framework system (based on low-level array-database concepts) uses MPI-based parallel processing techniques, storing and analyzing in memory and data distribution methods to improve the scalability of traditional OLAP systems. More specifically, it is based on a matrix-based storage model and a hierarchical data organization adopted to distribute scientific datasets on multiple nodes and, consequently, to enable efficient parallel data processing. Furthermore, the workflow management system, integrated into the Ophidia framework, makes the platform more flexible, helps reduce the complexity of scientific experiments by splitting them into simpler activities and increases reusability [Fiore et al., 2017].
The architecture is mainly composed of four layers [Elia et al., 2016]:

- **The I/O nodes layer** handles operations related to I/O activities on the underlying storage system. The internal storage model is designed to handle scientific n-dimensional data (datacubes), which are partitioned and distributed on multiple I/O servers and physically stored in groups of arrays (called fragments). To analyze the data characterizing this structure, approximately 100 functions (called primitives) based on array manipulation have been developed;
- **The Compute nodes layer** is responsible for the management and manipulation of datacubes, intended as a large collection of fragments, through a series of distribution functions, called operators;
- **The analytics workflow manager** implements an internal workflow runtime engine to submit and handle workflows of Ophidia operators, managing job queuing, submission and tasks dependencies;
- **The Front-End** exposes a set of server-side interfaces for the submission of Ophidia workflows and operators: WS-I, GSI / VOMS and OGC-Web Processing Service (WPS) are some examples.

The Ophidia software is available online as RPMpackages, source code and standalone virtual machine for fast deployment on desktop PCs (http://ophidia.cmcc.it/download/). In addition, a complete installation is available (upon registration) at CMCC premises to maximise Ophidia's analysis and datacube manipulation features in a multi-node environment (https://ophidialab.cmcc.it) (see section 4.5).

4.2 Datacube

A Datacube represents data along a certain measure of interest. Although called a "cube", it can be n-dimensional. Each dimension represents an attribute while the cube cells represent the values of the measured variable next to the considered dimensions. In Figure 1, we can see an example of a datacube.

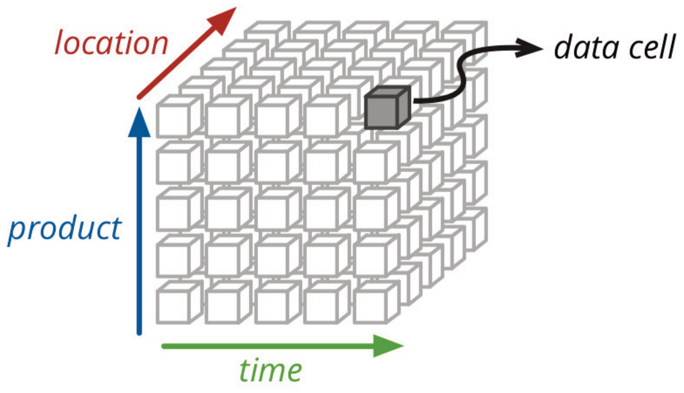


Figure 1 Datacube example.

It is an abstraction of data used in the evaluation of aggregated data from various points of view. A data cube can also be described as the multidimensional extension of two-dimensional tables. It can be considered as a collection of identical superimposed 2-D tables. Datacubes are used to represent data that are too complex to be described by a table of columns and rows.

4.3 Operators and primitives

The Ophidia framework provides support for multiple types of data and metadata management operations (both sequential and parallel There are currently more than 50 operators available who offer various types of functionality.
The main classes of operators include [Fiore et al., 2017]:

- **Data import/export**: to move data in/out of Ophidia. They often represent the first and last step in an end-user workflow;
- **Metadata management**: to manage metadata (e.g. data provenance) and enable search and discoverability of datasets
- **Reduction/aggregation**: to perform statistics on datacube values (e.g. max, min, average, sum, etc.);
- **Subsetting**: to slide/dice data based on a rich set of filters datacube dimensions;
- **Data exploration**: to explore the data stored in Ophidia through a rich set of filters;
- **Cube intercomparison**: to perform binary operations (e.g. sum, difference, etc.);
- **Execute applications**: to execute external applications (for example for post-processing or visualization);
- **Execution queries**: to execute user-defined queries that exploit the primitives available in Ophidia.

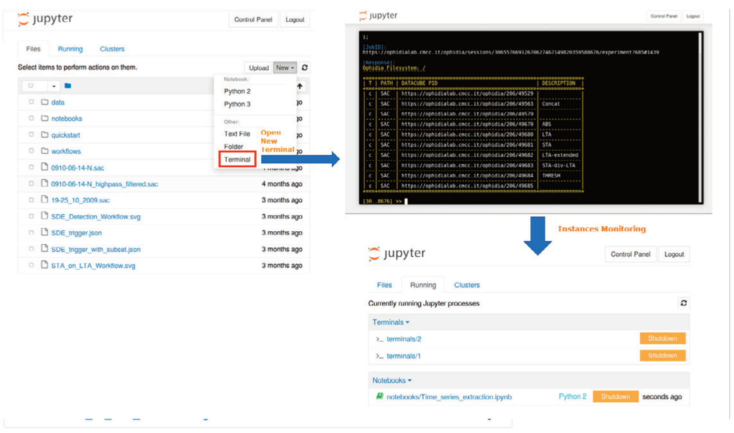
4.4 Client components for Ophidia

Ophidia provides two main client components to interact with the backend framework: the **Ophidia Terminal** (Oph_term) and the **PyOphidia** library.
The Ophidia Terminal is an advanced bash like interpreter through which it is possible to submit requests to an Ophidia server. A user can submit simple commands in an interactive and sequential manner as well as complex workflows, described by JSON files structured according to the Ophidia Workflow JSON schema. The Terminal provides the possibility to use

environment variables and aliases, keyboard shortcuts, recursive historical searches, implicit session management etc.
PyOphidia is a Python library (https://github.com/OphidiaBigData/PyOphidia) for interacting with a (remote) Ophidia server. It is an alternative to Oph_Term to submit workflow requests, Ophidia operators, or to develop your own application using the Python language. It runs on Python 2.7, 3.3, 3.4 and 3.5, it has no dependencies with other Python libraries and is pure-Python code. The latest version of PyOphidia is v1.5.0.

4.5 Ophidia Lab

OphidiaLab [OPHIDIALAB, 2017] is a scientific data analysis environment developed at the CMCC Foundation. It provides an environment that leverages a server-side approach and combines different data analysis tools to support researchers in their daily activities. OphidiaLab consists of several components: an Ophidia cluster, a JupyterHub instance including a set of pre-installed Python libraries for running data manipulation, analysis and visualisation, a data publication service and a tool for the infrastructure monitoring (mainly intended for administrators).
JupyterHub, a multi-user Hub, generates and manages multiple installations of a single user Jupyter notebook server. JupyterHub can be used to serve notebooks to a class of students, a corporate data or a scientific research group. Figure 2 shows some features of Ophidia Lab.



Select items to perform actions on them.

data	
notebooks	
quickstart	
workflows	
0910-06-14-N.sac	
0910-06-14-N_highpass_filtered.sac	4 months ago
19-25_10_2009.sac	3 months ago
SDE_Detection_Workflow.svg	3 months ago
SDE_trigger.json	3 months ago
SDE_trigger_with_subset.json	3 months ago
STA_on_LTA_Workflow.svg	3 months ago

```
i;

[JobID]:
https://ophidialab.cmcc.it/ophidia/sessions/38655766912678627467149828359588676/experiment?685#1439

[Response]:
Ophidia Filesystem: /

+===+======+=============================================+=============+
| T | PATH | DATACUBE PID                                | DESCRIPTION |
+===+======+=============================================+=============+
| c | SAC  | https://ophidialab.cmcc.it/ophidia/206/49529 |             |
| c | SAC  | https://ophidialab.cmcc.it/ophidia/206/49563 | Concat      |
| c | SAC  | https://ophidialab.cmcc.it/ophidia/206/49579 |             |
| c | SAC  | https://ophidialab.cmcc.it/ophidia/206/49679 | ABS         |
| c | SAC  | https://ophidialab.cmcc.it/ophidia/206/49680 | LTA         |
| c | SAC  | https://ophidialab.cmcc.it/ophidia/206/49681 | STA         |
| c | SAC  | https://ophidialab.cmcc.it/ophidia/206/49682 | LTA-extended|
| c | SAC  | https://ophidialab.cmcc.it/ophidia/206/49683 | STA-div-LTA |
| c | SAC  | https://ophidialab.cmcc.it/ophidia/206/49684 | THRESH      |
| c | SAC  | https://ophidialab.cmcc.it/ophidia/206/49685 |             |
+===+======+=============================================+=============+

[30..8676] >>
```

Currently running Jupyter processes

Terminals
- terminals/2
- terminals/1

Notebooks
- notebooks/Time_series_extraction.ipynb Python 2 seconds ago

Figure 2 Some OphidiaLab functionalities.

5. Procedure development to import SAC data into Ophidia

Ophidia has been developed in the context of climate change science, which uses the Network Common Data Form (**NetCDF**) as the main reference standard. NetCDF is a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data. [UNIDATA, 2017]. In this context, the Ophidia framework offers a specific operator (OPH_IMPORTNC) able to import NetCDF files into a datacube, allowing further analyses and manipulation through the available operators. To exploit the features of Ophidia on SAC datasets, it was necessary to develop a new operator. The following section describes the implementation steps of the **OPH_IMPORTSAC** operator.

5.1 Logical data model

The development of a new Ophidia operator is facilitated by the modularity of the code with a set of basic functions common to all operators. Furthermore, the process of fragmentation and distribution of imported data follows the same approach regardless of the format of the input dataset. Specifically, the Ophidia datacube structure requires at least two dimensions that identify the point to be imported, while SAC stores data as one-dimensional time series, i.e. a series of time-stamped data. A new dimension had to be defined to make the input data compliant with the Ophidia storage model. A new variable, called "block", has been defined: each block represents a set of samples corresponding to one hour. Figure 3 summarizes the logical organisation of SAC , following the introduction of the artificial "block" dimension.

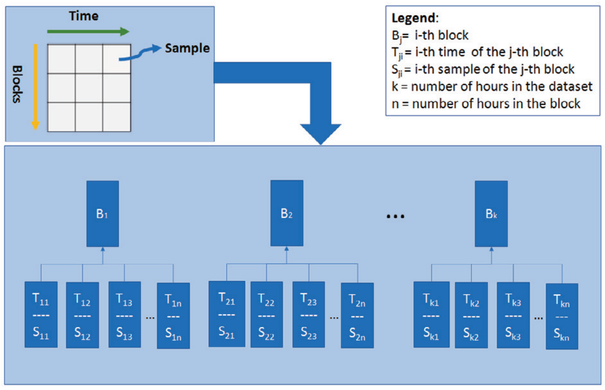


Figure 3 Logical data model.

The number of blocks is equal to the number of hours in the dataset and is stored in the variable of the SAC header called "F".

5.2 Development of library for reading data in SAC format

The Ophidia software modules are written in C language, which was then used to develop the SAC file reading library. We started using the *sacsubc.h* library, which is a component of the *Generic Seismic Application Coding* (GSAC) software [Herrmann, 2013]. The *readsaclib.h* library, developed in this work, is a generalization of the *sacsubc.h* library. It abstracts from the specifics of the SAC data format, offering simpler functions for implementing the OPH_IMPORTSAC operator, described below.
This section presents the prototypes of the main functions and data structures with a brief description.

The `headerDataInfo` data structure contains information extracted from the SAC header.

```
typedef struct headerData {
        char* filename;
        long int startDataPoint;
        unsigned long long int samplesNumber;
        unsigned long int blocksNumber;
        long int samplesForBlock;
        long int samplesForLastBlock;
        float samplingRate;
        struct tm *datetime;
} headerDataInfo;
```

The `startDataPoint` attribute stores the location of the file from which the samples start.

As we saw in diagram 1 of section 3.2, SAC data is stored from byte 632 (158 words * 4 bytes).

`samplesNumber` stores the number of samples in the file taken from the SAC header NTPS variable. Its type is **unsigned long long int** since a one-year dataset contains 31.5 billion samples, a number that can't be expressed with the type **unsigned long int** (limited to about 4 billion).

`blocksNumber` stores the number of blocks, the artificial variable described in the previous section. It is read from the SAC variable called “F”.

`samplesForBlock` stores the number of samples for each block. It is calculated using the formula `samplesNumber /blocksNumber`. Although it is a derived variable, it should be stored to avoid reloading it at every step of the import function.

`samplesForLastBlock` is set to 0 if `sampleNumber` is divisible by `blocksNumber`, otherwise it is set with the remainder of the division of the two variables. It gives us an indication that the last block is made up of fewer samples than the others.

`samplingRate` is the dataset sampling rate. It stores the SAC header variable called DELTA (float type).

struct `tm* datetime` memorizes the origin time (dataset start date and time). SAC stores this time in the individual date components: year, month, day, hour, minute, second, millisecond (see section 3.2), after reading the individual variables and creating a structure that represents the date and time using the type **struct** `tm`.

The function prototypes is detailed below.

int getHeaderInfo `(headerDataInfo*hdi,` **char**`* filename)` stores the variables read in the SAC header as well as some additional information, including the file path in the filename. It stores various header information in the data structure explained above. It uses various utility functions explained below.

int getYear() reads the SAC header variable NZYEAR (year of departure of the dataset).

int getJDay() reads the SAC header variable NZJDAY (day of departure of the dataset, expressed in Julian calendar).

int getHour() reads the SAC header variable NZHOUR (hour of departure of the dataset).

int getMinute() reads the SAC header variable NZMIN (minute of departure of the dataset).

int getSecond() reads the SAC header variable NZSEC (second of departure of the dataset).

int getMSecond() reads the SAC header variable NZMSEC (milliseconds of departure of the dataset).

void md_from_jday `(`**int** `year,` **int** `day_of_year,` **int**`* gdate)` converts the day of the year from Julian calendar to the Gregorian calendar.

int getMonth `()` returns the month of the year. Uses the results of the **md_from_jday** function.

int getMDay`()` returns the day of the month. Uses the results of the **md_from_jday** function.

void getDateTime`(`**struct** `tm* info)`: This function formats a date to be stored in the **struct** `tm` structure using the previously described utility functions.

long int getSacHeader`(`**FILE**`* fptr)` receives in input a file pointer to a SAC file. Reads the header variables and stores them in the data structure defined in the library. Gives 0 for success, otherwise the error number.

int getSacData`(`**FILE**`* fptr,` **float**`** data,` **unsigned long long int** `maxSamplesNumber)` receives in input the file pointer to a SAC file and stores the read data in an array of data. Verifies that the `maxSamplesNumber`is under the limit defined in `MAXSAMPLES`. In case of overflow, `maxSamplesNumber` is set to `MAXSAMPLES`.

float getSamplingRate`()` returns the sampling rate (number of samples per second). It trivially reads the DELTA variable of the SAC header.

unsigned long long int getSamplesNumber`()` returns the number of samples (the SAC header variable called NPTS)

unsigned int getBlocksNumber`()` returns the number of blocks, stored in the SAC variable “F”, into which the dataset will be divided. This value represents the second dimension for Ophidia.

5.3 OPH_IMPORTSAC operator development

The Ophidia import procedure uses a process of fragmentation and hierarchical distribution of data to subdivide the dataset and import it into several elementary units (called fragments). These fragments can be analyzed in parallel by MPI-based operators in a multi-node, multi-core environment. In particular, the fragmentation operation is carried out following one of the dimensions that identify the values to be imported: in the specific case of SAC data, each fragment will contain the values corresponding to a predetermined subset of Blocks, while the distribution phase will store each fragment on one of the I/ O nodes. As mentioned above, each fragment consists of a series of key-value pairs in which the key identifies the Block, while the value is an array (time series) containing the values of the samples belonging to that Block.
The import procedure is therefore a parallel operator in which each task is responsible for creating a set of fragments. Fragments (and therefore blocks) are identified and associated with each task. This is a preliminary part of the execution, independent of the format of the file to be imported. At this stage, each task composing the import operator is associated with an equal set of fragments created in this process, in order to balance the workload associated with each execution core.
In the case of SAC data, if N (set during the submission phase of the import operator) is the total number of fragments to be created, B is the total number of blocks (hours) in which the SAC file is divided and P is the number of parallel tasks with which the OPH_IMPORTSAC operator is launched:

- Each fragment will be composed of B/N blocks;
- Each execution task will manage N/P fragments;
- Each task will import the data corresponding to B/P blocks.

In the second phase of the execution, the importsac operator accesses the source file, reads the information (samples) to be imported and creates the resulting datacube. In particular, for each fragment, each task performs the following operations:

- extraction of the relative samples from the source file, in particular those corresponding to a pre-set number of hours (set of blocks);
- creation of the corresponding fragment;
- storing of the fragment within the datacube that represents the entire source SAC file.

The following excerpts of code (simplified to facilitate reading) represent the operations performed by the i-th task for reading data and creating the relative fragment.

```
for(i=0; i<blocksNumber; i++){

              beginReadingPoint=i*(hdi.samplesForBlock);
              task ( &hdi, beginReadingPoint,hdi.samplesForBlock, 10,1);

}
```

For each block, we calculate the starting point in the file from which data is read. It launches the i-task that will read a portion of the file.

```
int task(headerDataInfo* hdi, unsigned long int beginReadingPoint, unsigned long int
numberSamplesToRead, INT numberSubBlock, unsigned int fragmentNumber)

{
       float* data;
       inti=0,j=0;
       unsigned int samplesForSubBlock,samplesForSubLastBlock;

       if(numberSamplesToRead==0||numberSubBlock<=0||fragmentNumber==0)return
ERROR;
       samplesForSubBlock=numberSamplesToRead/numberSubBlock;
       samplesForSubLastBlock=numberSamplesToRead%numberSubBlock;

       FILE*filesac=fopen(hdi->filename, "rb");
       if (filesac==NULL){

              perror("filesac opening");
              return ERROR;

       }
       fseek(filesac,hdi-
>startDataPoint+(beginReadingPoint*(sizeof(float))),SEEK_SET);

              samplesForSubBlock=numberSamplesToRead/numberSubBlock;
       samplesForSubLastBlock=numberSamplesToRead%numberSubBlock;
       for (i=0; i<fragmentNumber; i++){
              if(samplesForSubLastBlock>0) numberSubBlock--;


              for (i=0;i<numberSubBlock; i++){
                    if(!getSacData(filesac, &data, samplesForSubBlock)){

        // The code to populate the datacube is inserted here in the final version.
                    memset(data, 0, samplesForSubBlock*sizeof(float));

                            free(data);
                    }else
                            printf("getasac problem\n");

            }
            if(samplesForSubLastBlock>0){

                    getSacData(filesac, &data, samplesForSubLastBlock);
        //The code to populate the datacube is inserted here in the final version.
                    memset(data, 0, samplesForSubLastBlock*sizeof(float));

                    free(data);
            }
        }
        fclose(filesac);
        return 0;

}
```

Unlike the operation of writing a file that requires special measures or specific libraries for simultaneous access to the destination file in case of parallel application, it is worth noting that when it comes to reading, each task involved can access and retrieve information from the input file using a different file descriptor: each task will create a specific portion of the output datacube by accessing its section of the SAC file and retrieving the subset of samples assigned to it.

The developed operator then allows you to import a generic SAC file and can be submitted to an Ophidia platform using the following command:

oph_importsac measure=<<variable_name>>;src_path=<<file sac path>>; container=<<container_name>>; dim=SAMPLE|BLOCK;nfrag=<<number_of_fragmentsto create>>;ncores=<<number of cores to be used>>;

in which:

- *measure:* name of the variable in the input file to be imported: for the SAC format, it can be identified with groundvelocity;
- *src_path:* represents the path of the SAC file to be imported;
- *container:* represents an entity used in Ophidia to collect all the datacubes share the same structure: if not indicated, it is created automatically;
- *dim* represents the list (separated by |) of the dimensions that identify the samples: in the case of SAC, they are BLOCK and SAMPLE, as previously described;
- *nfrag* represents the number of fragments you want to create; using the nhosts parameter, you can also indicate it is also possible to indicate the number of hosts (I / O nodes) you want to use to store the output datacube (if not expressed, by default the system will use all available nodes);
- *ncores* is the number of cores (tasks) that will be used during the operator run.

5.4 Importing data and comparative performance tests

The OPH_IMPORTSAC operator allows importing the SAC format datasets into the Ophidia framework. A preliminary step is required to add the new dimension “block” to the SAC input file (variable “F” in the SAC header) as follows:
Run the SAC program from the command line and then run:

```
SAC> READHDR <<sac file name>>
SAC> CHNHDR F <<blocks number>>
SAC> WRITEHDR <<sac file name>>
```

The prepared file is uploaded to the Ophidia server.
After opening the Ophidia Terminal, the new OPH_IMPORTSAC operator is launched, as shown in the previous section.
Finally, a simple test was performed to calculate the Fast Fourier Transform (FFT) on a dataset recorded by the seismometer installed on a seafloor observatory. The FFT calculation was chosen because it is computationally costly. Below are some technical details of the test:

Dataset used for the test:
The dataset was a 10-day time series, consisting of about 600 million samples stored in a 648 Mb file.

SAC test environment:
Hardware: Intel(R) Core (TM) i5-3470 CPU @ 3.20GHz, 3201 Mhz, 4 core 8 Gb RAM
OS: Linux Mint 16 64 bit

Ophidia test environment:
CPU Xeon E5-2670, Nrcpu: 2, Nr core/cpu: 8, Clock Rate: 2,6 GHz, RAM: 64 GB
Cluster: configured with one node

Results

The FFT calculation was carried out using SAC and Ophidia and the two results were compared. SAC took about two minutes to finish the test, while Ophidia took about 14 seconds.

This first simple test, which represents a comparison between the SAC on a single PC analysis and the new one using Ophidia, provides experimental evidence of the greater efficiency of the Ophidia approach.

Conclusions and future steps

Starting from a case study, we described how our research group at INGV could benefit from the solutions proposed within the INDIGO project, in particular Ophidia, a big data analytics framework.

We wanted to apply Ophidia in the analysis of seismic data and the first challenge we encountered was importing the data in the SAC format into Ophidia. To apply Ophidia on SAC data, we had to act both at the application level and the internal operating logic. As a result, a new operator was developed to import the seismological data into Ophidia. A performance test, based on the application of FTT on a long time series, has shown that the Ophidia framework is more efficient than the analysis tools normally used by our research group.

As the next step, we want to develop a new operator to export Ophidia datacubes in the SAC output format. We would also like to further exploit Ophidia's capabilities to analyse seismological data by implementing algorithms (such as, automatic seismic event detection) and creating ad hoc operators (e.g. exportSAC operator).

Acknowledgements

Thanks to Robert B. Herrmann, professor of Geophysics at Saint Louis University, for authorizing the use of the *sacsubc.h* library in this work.

Sections of this paper were presented at the INDIGO Summit 2017 international conference hosted in Catania from May 9 to 12 2017 under the title **Using Ophidia in EMSO** [Marcucci et al., 2017].

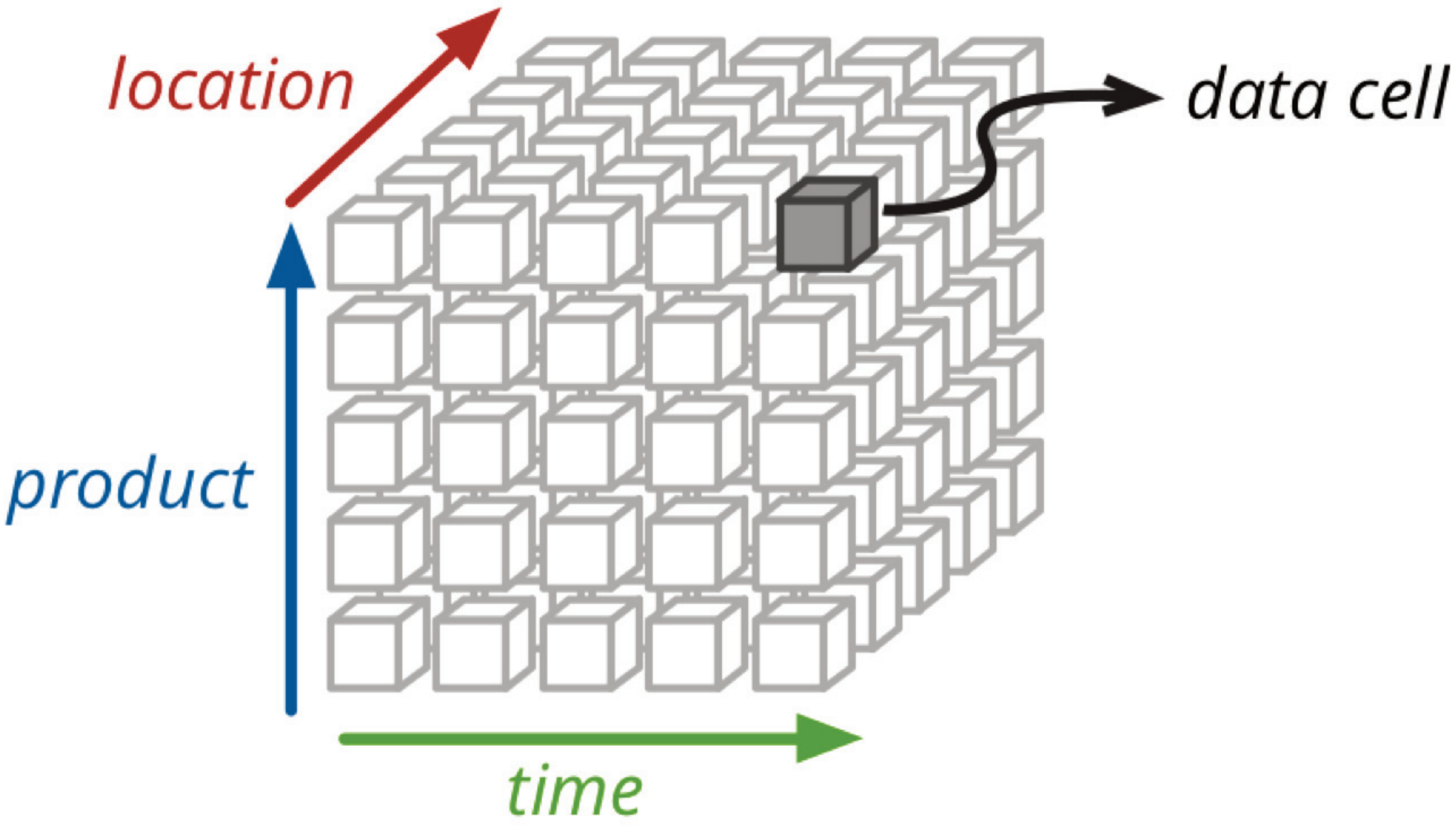
We thank Dr. Maja Giljanovic for proof-reading the manuscript.

References

INDIGO-DataCloud, https://www.indigodatacloud.eu/
EMSO-ERIC, www.emso.eu
MOIST, http://www.moist.it
IRIS WEBSITE, http://ds.iris.edu/ds/nodes/dmc/software/downloads/sac/
SAC DATA FORMAT, https://ds.iris.edu/files/sacmanual/manual/file_format.html
Fiore S., Palazzo C., D'Anca A., Elia D., Londero E., Knapic C., Monna S., Marcucci N.M., Aguilar F., Pł.ciennik M., De Lucas J.E., (2017). Big Data Analytics on LargeScale Scientific Datasets in the INDIGODataCloud Project. In: Proceedings of the Computing Frontiers Conference 2017 May 15 (pp. 343348), ACM, doi:10.1145/3075564.3078884
OPHIDIA, http://ophidia.cmcc.it/
Fiore S., D'Anca A., Palazzo C., Foster I.T., Williams D.N., and Aloisio G., (2013). Ophidia: Toward big data analytics for escience. In: Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June 2013, pp. 2376–2385, Elsevier.
Elia D., Fiore S., D'Anca A., Palazzo C., Foster I., Williams D.N., and Aloisio G., (2016). An inmemory based framework for scientific data analytics. In: Proceedings of the ACM International Conference on Computing Frontiers, pp. 424–429, ACM 2016.
UNIDATA, NetCDF, https://www.unidata.ucar.edu/software/netcdf/
Herrmann R.B., (2013). Computer programs in seismology: An evolving tool for instruction and research, Seism. Res. Lettr., 84, 10811088, doi:10.1785/0220110096
OPHIDIALAB, https://ophidialab.cmcc.it
Marcucci N.M., Monna S., D'Anca A. & Fiore S., (2017). Using Ophidia in EMSOINGV. Oral Presentation at INDIGO SUMMIT 2017 Conference (Catania, Italy).

Coordinamento editoriale e impaginazione
Francesca DI STEFANO, Rossella CELI
Istituto Nazionale di Geofisica e Vulcanologia

Progetto grafico e impaginazione
Barbara ANGIONI
Istituto Nazionale di Geofisica e Vulcanologia



ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA