Towards High Performance Data Analytics for Climate Change

Sandro Fiore[1(✉)], Donatello Elia[1,2], Cosimo Palazzo[1], Fabrizio Antonio[1], Alessandro D'Anca[1], Ian Foster[3], and Giovanni Aloisio[1,2]

[1] Euro-Mediterranean Center on Climate Change Foundation, Lecce, Italy
sandro.fiore@cmcc.it
[2] University of Salento, Lecce, Italy
[3] University of Chicago & Argonne National Laboratory, Chicago, USA

Abstract. The continuous increase in the data produced by simulations, experiments and edge components in the last few years has forced a shift in the scientific research process, leading to the definition of a fourth paradigm in Science, concerning data-intensive computing. This data deluge, in fact, introduces various challenges related to big data volumes, formats heterogeneity and the speed in the data production and gathering that must be handled to effectively support scientific discovery. To this end, High Performance Computing (HPC) and data analytics are both considered as fundamental and complementary aspects of the scientific process and together contribute to a new paradigm encompassing the efforts from the two fields called High Performance Data Analytics (HPDA). In this context, the Ophidia project provides a HPDA framework which joins the HPC paradigm with scientific data analytics. This contribution presents some aspects regarding the Ophidia HPDA framework, such as the multidimensional storage model, its distributed and hierarchical implementation along with a benchmark of a parallel in-memory time series reduction operator.

Keywords: HPDA · Climate change · Scientific data analysis · Storage model · Multidimensional data

1 Introduction

Scientific research has been experiencing a shift over the last few years due to the enormous increase of data produced by simulations, experiments, edge components, etc. which has led to the definition of a fourth paradigm in Science concerning data-intensive computing [7]. The deluge of data, however, poses several challenges that must be tackled accordingly to cope with bigger data volumes, heterogeneous formats and different frequency in data generation. To fully support data-intensive scientific applications, High Performance Computing (HPC) and data analytics are both deemed as fundamental and complementary aspects of the scientific process [21], providing together a new paradigm for

M. Weiland et al. (Eds.): ISC 2019 Workshops, LNCS 11887, pp. 240–257, 2019.
https://doi.org/10.1007/978-3-030-34356-9_20

eScience named High Performance Data Analytics (HPDA). Related to that, in the current scientific landscape, the Ophidia project provides a HPDA framework joining HPC paradigms with scientific data analytics approaches. This paper describes key aspects of the Ophidia HPDA framework, such as the multidimensional storage model design and the related distributed and hierarchical implementation across multiple, heterogeneous physical resources; additionally, it also presents a benchmark of a key analytical operator for parallel, in-memory time series reduction.

The rest of this paper is organized as follows. After the review of some key challenges for scientific data analysis in Sects. 2 and 3 presents some of the main aspects of the Ophidia project like its internal storage model and partitioning/distribution schema design and implementation. Section 4 presents a time series reduction benchmark, while Sect. 5 presents state of the art projects in the scientific data analytics area. Finally, Sect. 6 draws the main paper conclusions and describes future work.

2 Main Challenges

By reviewing a previous work presented at BDEC [2], we summarize some key challenges [1,3,10,18] for addressing scientific data analytics at scale. In particular, we (i) discuss the need to review the *scientific data analysis workflow* (*shifting from client to server-side approaches*), and (ii) highlight *storage, data management and metadata-related challenges.*

The workflow commonly used in production for scientific discovery has traditionally been based on the *search, locate, download and analyze* steps, typically performed on a researcher's desktop. Such workflow could not scale for several reasons including (i) ever-larger scientific datasets, (ii) time- and resource- consuming data downloads, and (iii) increased problem size and complexity requiring bigger computing facilities. Server-side approaches have helped address challenges related to the analysis of very large datasets, keeping data produced from simulations or gathered from observations close to the data center facilities.

As regards the *storage*, new storage models are essential to support datacube-oriented analytics. New data organizations are needed to better fit the intrinsic datacube model of n-dimensional data. Data partitioning and distribution can enable parallelism, whereas indexing, replication, and caching can improve execution efficiency and throughput. In a multidimensional space, dimensional data independence - where the storage model should be independent of the number of dimensions - should be clearly addressed to provide the right level of separation of concerns.

Finally, *metadata* represents a valuable source of information for data description, data search & discovery as well as for properly (from a semantics perspective) running scientific tasks. In this regard, with the shift towards server-side approaches, it becomes extremely relevant to support, in close proximity to the compute and data storage facilities available in a data center: (i) metadata management capabilities to index datasets and support datasets search & discovery,

(ii) provenance metadata capture and collection for describing the flow of analytics operators in scientific data analysis experiments; (iii) integration of information linking cross-related digital objects (using PIDs), (iv) the creation of new community-oriented tools to enrich metadata and provide, at the same time, a way to move this process towards much more open, multi-level and collaborative forms, targeting Open Science-oriented approaches.

3 The Ophidia Project

In the eScience landscape, the Ophidia project [11,12] provides a HPDA framework joining HPC paradigms with scientific data analytics approaches. Primarily exploited in the climate domain, it provides a domain-agnostic architectural design, which makes it suitable for any scientific domain dealing with multidimensional data formats [15]. A complete architectural overview of Ophidia is reported in [14].

Ophidia provides in-memory, parallel, server-side data analysis & I/O, an internal storage model and a hierarchical data organization to manage large amounts of multidimensional scientific data. The multidimensional storage model and related implementation aspects are presented in Sect. 3.1.

Array-based functionalities ("primitives") as well as datacube kernels ("operators") represent two levels through which end-users can operate on scientific data performing analytics tasks. Such aspects are discussed in detail in Sect. 3.2.

3.1 Multi-dimensional Storage Model

The objective of addressing efficient climate data management inherently leads to the key challenges of properly dealing with scientific multidimensional data.To achieve this goal, Ophidia implements a storage model leveraging the datacube abstraction from the well-known On-Line Analytical Processing (OLAP) systems in the databases field.

In such a context, a datacube consists of several measures representing numerical values that can be analyzed over the available dimensions. The multidimensional data model exists in the form of star, snowflake or galaxy schema. The Ophidia storage model [14] (see Fig. 1) builds on top of the classic star schema. In such a schema, the data warehouse implementation consists of a large central table (the *fact* table) that contains all the data and a set of smaller tables (*dimension* tables), one for each dimension. The dimensions can also define *hierarchies*, which represent a convenient way to organize the information according to their level of aggregation; a very common example is the time dimension, which can represent information at various levels of aggregations (e.g. *hours* ->*days* ->*months*, etc.).

In Fig. 1, the fact table is represented with the Dimensional Fact Model (DFM) [17], a conceptual model for data warehouse (see Fig. 1a). The example shows one fact table (FACT) with four dimensions (dim1, dim2, dim3, and dim4), where the last dimension is modeled through a 4-level concept hierarchy (lev1,

lev2, lev3, lev4), and a single measure (measure). This schema can be easily used to map a NetCDF file produced, for example, by a global climate simulation, where the four dimensions correspond to *latitude, longitude, depth,* and *time,* while the measure can represent the *air temperature*. The classic Relational-OLAP (ROLAP) logical model can then be used to implement the star schema (see Fig. 1b).

In terms of storage model, Ophidia implements a two-step-based evolution of the star schema. The first step introduces the support for array-based data types (see Fig. 1c), by merging multiple rows into a single binary array. Rows are merged according to one or more dimensions. In this way, an array contains the values of the measure related to all the possible configurations of these dimensions. The second step performs the mapping of the set of foreign keys (*fks*) (see Fig. 1d), related to the remaining subset of dimensions, to a single new key. Thus, a multidimensional array can be managed using single tuple (e.g., an entire time series) identified by one key (a numerical ID). It is worth noting that, thanks to this second step, the Ophidia storage model is *independent of the number of dimensions*, unlike the classic ROLAP-based implementation. Hence, the system implements a key-value schema (i) supporting n-dimensional data management, (ii) exhibiting data locality, and (iii) reducing disk space occupancy.

A combination of values of m dimensions ($m < n$) is mapped through a numerical function onto the key attribute: $ID = f(fk\_dim_1, fk\_dim_2, \ldots, fk\_dim_m)$; the corresponding dimensions are defined in Ophidia as *explicit dimensions*. The array attribute manages the other *n-m* dimensions, called *implicit dimensions*.

The ID key is defined as a sequential integer positive number computed with the following function (1).

$$ID = \sum_{j=i}^{m} S_j fk_j, \; with \; S_j = \prod_{i=j+i}^{m} size(d_i) \; \forall \; j = 1, \ldots, m-1 \; and \; S_m = 1 \quad (1)$$

where m is the number of explicit dimensions, $(fk_1, fk_2, \ldots, fk_m)$ is the particular configuration of dimension indexes and $size(d_j)$ is the size of the *j-th* dimension.

In our example, latitude, longitude and depth could be the explicit dimensions, whereas time would be the implicit one (in this case 1-D array). The mapping onto the Ophidia key-array storage model would therefore result in a single table with two attributes:

- an ID attribute: $ID = f(fk_{latitudeID}, fk_{longitudeID}, fk_{depthID})$ as a numerical data type;
- an array-based attribute, managing the implicit dimension time, as a binary data type.

In terms of implementation, several traditional RDBMSs allow data to be stored as a binary data type, by exploiting for instance the string data type

(as CHAR, BINARY, BLOB, TEXT types), but they do not provide a way to manage the array as a native data type. The reason is that the available binary data type does not look at the binary array as a vector, but rather as a single binary block. Therefore, we have designed and implemented a comprehensive set of array-based primitives to manage the arrays stored in Ophidia according to its internal storage model.

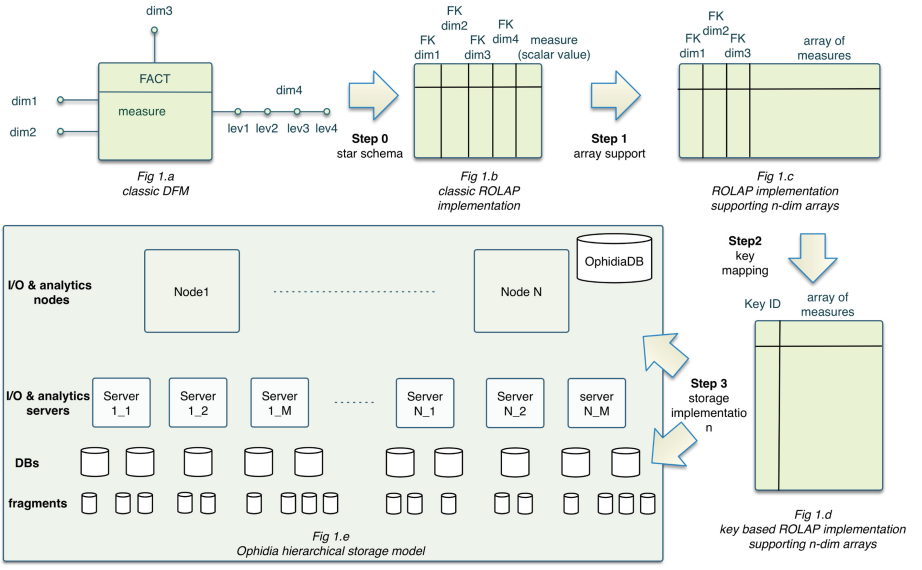

Fig. 1. Ophidia storage model and implementation

With regard to the physical mapping onto the storage systems (see Fig. 1e), Ophidia horizontally partitions this very long table into several *fragments* to efficiently handle big datacubes on the physical file system. This fragmentation is driven by the underlying resources following a hierarchical approach composed of four different levels, as shown in Fig. 1:

- Level 0: multiple Ophidia I/O & analytics nodes (multi-host);
- Level 1: multiple instances of Ophidia I/O & analytics servers on the same node (multi-server);
- Level 2: multiple instances of databases on the same IO & analytics server (multi-DB);
- Level 3: multiple fragments on the same database (multi-table).

The total number of fragments associated with a datacube is the product of these four parameters, which represents key settings for fragments distributions. Fine tuning of these parameters is out of the scope of this paper and will be discussed in more detail in a future work.

Finally, from an end-user perspective, the logical/virtual file system hosting the datacube objects is defined as a *cube space* (see Fig. 2) in Ophidia (as opposed to the physical file system associated with files) due to the datacube abstraction delivered to the users. Related to this, there is a set of operators that specifically allows copying, moving, deleting and listing datacubes and folders in the cube space. Metadata information (like that stored in the header of scientific data, e.g. NetCDF) is separately stored into the OphidiaDB, i.e., the Ophidia system catalog.

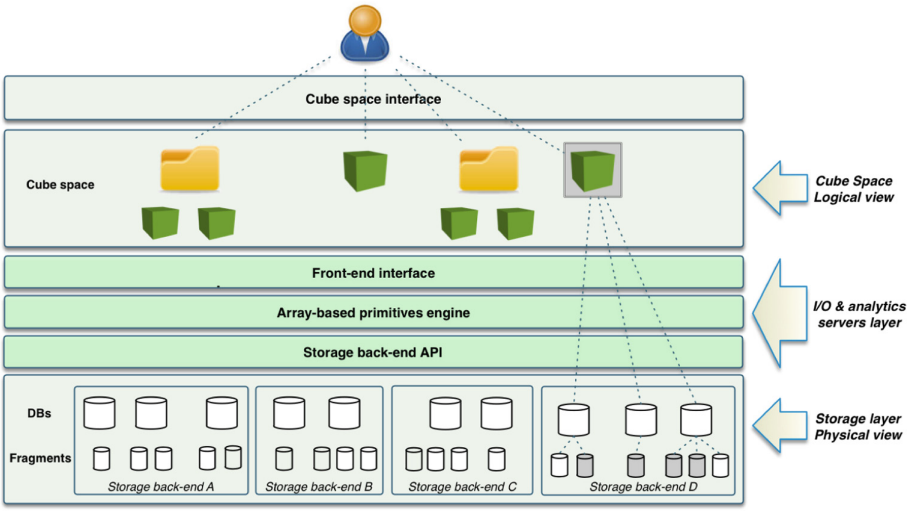

Fig. 2. Cube space abstraction and physical storage implementation/mapping

3.2 Array-Based Primitives and Parallel Operators

At the I/O and analytics server level (see Fig. 2), Ophidia provides an array-based engine and a set of array-based primitives as Structured Query Language (SQL) extensions relying on the User-Defined Functions (UDF) approach. About 100 primitives have been implemented. Among others, the available array-based functions allow the performance of data sub-setting, data aggregation (i.e., max, min, avg), array concatenation, algebraic expressions and predicate evaluation. We note that multiple plugins can be nested to implement a single and more complex task (e.g., aggregating by sum a subset of the entire array). Bit-oriented plugins have also been implemented to manage binary datacubes. The array-based processing is performed by the I/O and analytics servers engine, which supports the management of n-dimensional array structures both at the data type and primitives levels. An in-depth view of the array-based primitives has been discussed in a previous work [14].

Additionally,

- through the storage back-end API, the I/O and analytics servers can transparently interface to different storage back-ends such as POSIX-like file system, object stores, relational DBMSs and memory. As the main focus of Ophidia is the support of in-memory analytics, the default 'storage back-end' is *memory*.
- on GPU-equipped nodes, the I/O and analytics servers can run some primitives available as a CUDA-based implementation.

From a datacube abstraction perspective, the Ophidia HPDA platform provides several MPI-based parallel operators to manipulate (as a whole) the entire set of fragments associated with a datacube. Some relevant examples include: (i) data sub-setting (slicing and dicing), (ii) data aggregation, (iii) array-based primitives (the same operator can run all the implemented UDF extensions), (iv) datacube duplication, (v) datacube pivoting, (vi) NetCDF-import and export. Still, some metadata-oriented operators are also available to manage the cube space objects and provide the scientific metadata of a data cube or its partitioning/distribution parameters. Table 1 summarizes the main types of operators, by classifying them based on their primary feature and type of processing; as it can be noted, metadata operators are sequential, as opposed to data operators, which are all available in a parallel implementation (multi-process/multi-thread based) to take advantage of the storage-level partitioning and data distribution.

It is worth mentioning that despite the difference in terms of functionalities, the parallel operators are all executed in a similar fashion, exploiting the same runtime execution model (more information about this can be found in [12]).

4 Benchmark and Experimental Results

This section describes a benchmark defined for the performance evaluation and scalability of the Ophidia framework when running a typical data reduction analysis operator (*OPH_REDUCE2*). Although Ophidia provides several classes of operators, as shown in Table 1, the *OPH_REDUCE2* is one of the most used and interesting in terms of parallel data processing; it addresses array-based data reduction to compute statistical indicators such as, among others, maximum, minimum, average and standard deviation.

In this respect, the computation of the average value of the time series for each point in a 3-dimensional spatial domain (e.g. latitude, longitude and pressure level) is considered, under different data partitioning and distribution settings. While previous works have addressed coarse-grain, application-level and end-user perspective scenarios by primarily focusing on scientific use cases (i.e., workflows) for climate indicators [9] and multi-model analysis [13], the goal of the benchmark proposed in this paper is to provide key insights into largely used core Ophidia operators. In this respect, the *OPH_REDUCE2* is one of the best candidates because it relates to statistical analysis, which is very common in any scientific data analysis.

Table 1. Main Ophidia operator classes

Class	Processing type	Operator(s)
I/O	Parallel	-Data import (OPH_IMPORTNC, OPH_IMPORTFITS) -Data export (OPH_EXPORTNC) -Append data from files to datacubes (OPH_CONCATNC) -Generate random data (OPH_RANDUCUBE)
Time series processing	Parallel	-Apply generic time series transformation (OPH_APPLY)
Datacube reduction	Parallel	-Reduction over the implicit dimensions (OPH_REDUCE) -Reduce time series based on concept hierarchies (OPH_REDUCE2) -Reduction over the explicit dimensions (OPH_AGGREGATE)
Datacube subsetting	Parallel	-Subset data based on dimension coordinates or indexes (OPH_SUBSET)
Datacube combination	Parallel	-Compare different cubes (OPH_INTERCUBE) -Merge multiple cubes (OPH_MERGECUBES)
Datacube structure manipulation	Parallel	-Split the fragments (OPH_SPLIT) -Merge fragments together (OPH_MERGE) -Change dimension order/type (OPH_ROLLUP, OPH_DRILLDOWN, OPH_PERMUTE)
Datacube/file system management	Sequential	-Delete a datacube (OPH_DELETE) -Manage virtual file system (OPH_FOLDER) -Browse real file system (OPH_FS)
Metadata management	Sequential	-Metadata management (OPH_METADATA) -Provenance exploration (OPH_CUBEIO) -Datacube info (OPH_CUBESCHEMA)
Datacube exploration	Sequential	-Datacube exploration (OPH_EXPLORECUBE) -File exploration (OPH_EXPLORENC)

It should be noted that the benchmark is not intended to be large-scale per se, but rather to provide some key insights into the scalability of the *OPH_REDUCE2* operator from a performance standpoint on a Terabyte-scale. The large-scale scientific data analysis scenario that can be targeted with Ophidia is not about running a single operator on a Petabyte-scale datacube, but rather several operators in the same analysis (i.e., workflows) on hundreds/thousands of Terabyte-scale datacubes. In this respect, the Terabyte-scale performance metrics addressed in the paper turn out to be very good indicators of the scalability of a core Ophidia data reduction operator.

4.1 Benchmark Definition

As stated above, the main focus of this benchmark is to evaluate the scalability of data reduction operations. In particular, the *OPH_REDUCE2* parallel operator has been tested to compute the average value of each time series stored in the input datacube.

An example of the Ophidia command used to run the *OPH_REDUCE2* operator is described as follows (more details about the *OPH_REDUCE2* operator available options can be found on the online Ophidia documentation[1]):

```
oph_reduce2 operation=avg;dim=time;ncores=10;nthreads=10;
    cube=<input_cube>;
```

The declarative statement presented above shows: (i) the type of reduction operation (*avg*); (ii) the reduction dimension (*time*); (iii) the number of MPI processes (set to 10 *ncores*); (iv) the number of threads for each process (set to 10 *nthreads*); and (v) the input datacube (*input_cube*).

In this benchmark, three tests have been identified and set up to evaluate the behaviour of Ophidia under different settings, with the aim of providing multiple insights from different perspectives.

1. *Strong scalability.* This test case aims to evaluate the platform scalability by measuring the *OPH_REDUCE2* execution time on a fixed problem size while increasing the number of executed parallel tasks, from 1 to 100.
2. *Weak scalability.* In this test, the *OPH_REDUCE2* operator is executed by scaling up the data size along with the number of used parallel tasks, from 1 to 100; only the data size for a unit of computation is fixed.
3. *Array-oriented.* This test aims to evaluate the performance of the framework while increasing the length of the binary array, with fixed data partitioning and number of parallel tasks.

It is worth mentioning that, in order to better adapt the data size used for the experiments, while trying to maximize the amount of memory used from the environment, the input datacubes have been derived from random data. The *OPH_RANDCUBE2* operator has been used to manually tune the datacube structure and populate it with random values (generated through a first order autoregressive model). The maximum problem size used is actually slightly different in the various tests due to datacube structure and fragmentation requirements. An example of the *OPH_RANDCUBE2* statement used during the tests to generate a random-data datacube is:

```
oph_randcube2 exp_dim=lat|lon|plev;exp_dim_size
    =1200|800|24;imp_dim_size=11680;imp_dim=time;measure=
    tas;nfrag=240;ntuple=19200;nhost=5;algorithm=
    auto_reg_first_order;
```

Such command is very flexible, providing the end users with multiple options to model and create a datacube. In this case, a 4d (*lat, lon, plev, time*) datacube is created with a total number of $1200 \times 800 \times 24 \times 11680$ tas elements (*float* type), stored in 240 (*nfrag*) fragments, each one storing 1.92×10^4 (*ntuples*) tuples and distributed across 5 (*nhost*) nodes running the native in-memory I/O & analytics servers. The first three dimensions (*lat, lon, plev*) are the explicit

[1] OPH_REDUCE2 documentation http://ophidia.cmcc.it/documentation/users/operators/OPH_REDUCE2.html.

ones (*exp_dim*) in the datacube, whereas time is the implicit or array-based one (*imp_dim*); tas is the name used to refer to the climate air temperature variable.

The main metrics measured during the tests include the *execution time* and the *data size*; other metrics derived from these include:

- *efficiency* is the percentage rate of the sequential execution time over the parallel execution time divided by the computation units (in the *strong scalability* test) or the rate of sequential execution time with respect to the parallel time (in the *weak scalability* test);
- *processing throughput* is the rate of input data processed with respect to the execution time and it is measured as GB/s.

4.2 Test Environment

The benchmark has been performed in a real data center setting, on a dedicated cluster designed for in-memory analytics, hosted at the CMCC SuperComputing Centre in Lecce (Italy).

The cluster is composed of five fat nodes, individually equipped with 256 GB of main memory, 1TB of local disk and 2 Intel Xeon processors (2×10 cores), for a total of 100 physical cores. These nodes are used for the execution of the native Ophidia in-memory I/O & analytics servers.

The storage is shared across the five nodes, which are connected together through a high-speed network (10Gb/s). The shared storage exploits a GlusterFS file system[2] distributed over five disks with about 60TB of total raw disk capacity. The official 1.4.0 release of Ophidia has been deployed on the cluster.

4.3 Experimental Results and Discussion

Multiple runs of the *OPH_REDUCE2* operator have been executed for each configuration of the three tests and the average time has been considered in the results hereafter. It is important to mention that all average values feature a 95% confidence interval whose maximum relative error is at most 7%.

Strong Scalability. In this first test case, the datacube size has been fixed to about 1TB, consisting of approximately 2.7×10^{11} floating point values organized into 2.3×10^7 time series (1.17×10^4 elements each one). At the storage level, the data has been partitioned into 1200 fragments evenly distributed over the five I/O & analytics servers, each one running on a single node and managing around 200 GB of data/node. The partitioning parameters have been defined to ensure that the number of fragments processed by each task is always well balanced while scaling up the number of parallel tasks from 1 to 100.

[2] GlusterFS documentation https://docs.gluster.org/en/latest/.

The size of the resulting output datacube is approximately 90 MB (18 MB/node, 4 order of magnitude smaller than the input datacube, but with the same degree of fragmentation). Table 2 shows a summary of the results obtained from this test, which include speedup, efficiency and data reduction throughput.

Table 2. Summary of results for the strong scalability test. In this case, the data size is constant (1TB) as well as the partitioning parameters (1200 fragments, 5 nodes, 1 I/O & analytics server/node).

Number of tasks	Execution time [s]	Efficiency [%]	Throughput [GB/s]
1	1290.8	100	0.8
10	144.3	89.4	6.9
20	73	88.5	13.7
40	35.5	90.8	28.2
60	23.4	91.8	42.9
80	19.5	82.7	51.4
100	17.6	73.2	56.8

The first row of the table provides the results of a sequential computation, while the others show the results for parallel processing. The number of tasks has been increased up to 100 to show the behaviour of the operator up to the full utilization of the cluster resources. Figure 3a provides a graphical representation of the efficiency and data reduction throughput (GB/s).

As it can be clearly seen, the efficiency is stable around 90% up to 60 tasks, going down to 73% only when the cluster resources are fully utilized. The best case exhibits an execution time of 17.6 s, which corresponds to 56.8 GB/s of processed data. Finally, besides the analytics performance evaluation, the benchmark shows how the distributed storage model implementation efficiently scales up with the problem size over multiple nodes, allowing to tackle larger-scale scenarios. This is actually something that would not be feasible (from a scalability standpoint) on a single host.

Weak Scalability. In this second test, the data size has been increased along with the number of tasks used to run the data reduction (average) operation. With respect to the previous test, since the data size and partitioning change over the different runs, the number of fragments per task has been fixed to 1 to make the results evaluation easier. Each fragment contains about 2.8×10^9 floating point values organized into 2.4×10^5 elements time series (1.17×10^4 elements each) for a total of 10.4 GB of data.

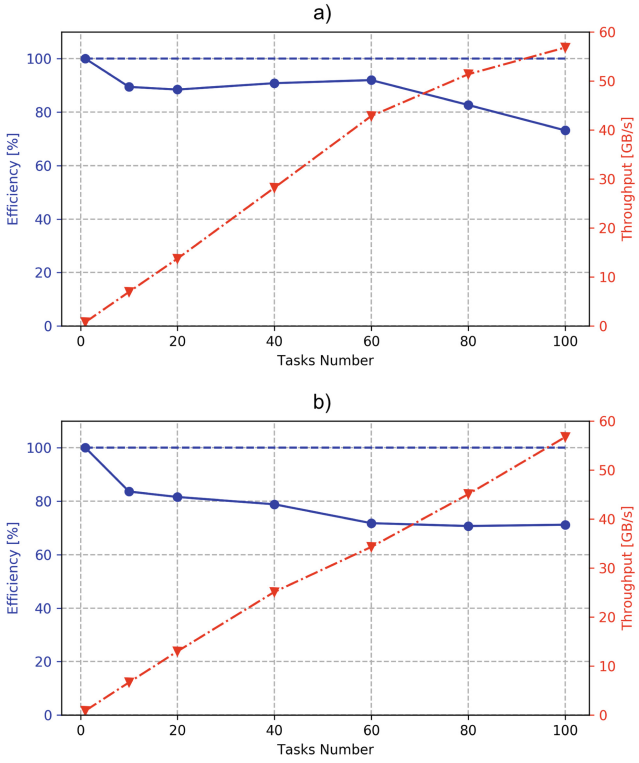

Fig. 3. Results of the strong (a) and weak (b) scalability tests. Efficiency is plotted with the full line, while data reduction throughput is plotted with the dashed line. The straight dashed line represents the ideal efficiency.

The *OPH_REDUCE2* operator has been executed by scaling up the number of fragments together with the parallel tasks up to 100 (i.e., 1.04 TB). In this test, the number of servers has been increased together with the data size. In particular, a single I/O & analytics server has been used for the first three configurations (i.e., 1, 10, 20), while 2, 3, 4 and 5 servers have been used for the other configurations. Through this setup, it has been possible to exploit the full resources provided by the cluster.

Table 3 provides an overview of the results of this test. The execution time and the total problem size have been measured for each run; based on them, we also inferred efficiency and processing throughput metrics.

In this case, as also highlighted in the plot in Fig. 3b the efficiency remains over 80% when the executed task remains bounded to one node, whereas it slightly decreases down to 70% when the computation runs over multiple nodes. This is related to the additional overhead required to manage a larger set of tasks running on multiple nodes. This experiment shows how the processing exhibits a good level of scalability over multiple nodes thanks also to the underlying distributed storage model and partitioning schema (the drop in performance

Table 3. Summary of results for the weak scalability test. In this case, the data size per task is constant (10.4 GB) while the number of tasks and fragments is scaled up from 1 to 100.

Tasks (fragments)	I/O & Analytics nodes	Execution time [s]	Efficiency [%]	Throughput [GB/s]	Data size [GB]
1	1	13.1	100	0.8	10.4
10	1	15.7	83.6	6.7	104.4
20	1	16.1	81.6	13.0	208.9
40	2	16.6	78.8	25.1	417.7
60	3	18.3	71.8	34.4	626.6
80	4	18.5	70.7	45.1	835.4
100	5	18.4	71.2	56.7	1044.3

is about 10% from 1 to 5 nodes, which corresponds to 2.3 s in our test). The execution time remains between 13 and 18.5 s, while the processing throughput peaks to almost 57 GB/s. The scalability of the storage model implementation is further demonstrated in the plot (see Fig. 3b), which highlights that the efficiency does not degrade as more resources are added; after the initial slowdown, it actually remains stable to around 70%.

Array-Oriented. In this test, the data partitioning parameters, as well as the number of tasks, have been kept constant while increasing the number of elements stored in the binary arrays (together with the total problem data size). Similarly to the previous test, the number of fragments per task has been set to one, hence the data has been split into 100 fragments, consisting of 2.3×10^5 time series each. Again, five nodes individually running an Ophidia in-memory I/O and analytics server have been exploited for this test. The execution time of the *OPH_REDUCE2* operator has been computed with a fixed number of tasks (i.e., 100), while increasing the array length by one order of magnitude each time, from 12 to 1.2×10^4 values (i.e., 1.03TB). The data reduction throughput has also been computed for each array configuration. Table 4 reports a summary of the results for this test.

The results confirm the benefits of the array-based organization in the Ophidia storage model implementation (which inherently takes full advantage of the data locality) from a processing performance standpoint. As it can be seen from Table 4, the throughput can be greatly improved, with the same amount of resources, just by increasing the array length.

To sum up, the results show that data partitioning and distribution jointly with the parallel processing approach implemented in Ophidia provide good performance under both strong and weak scalability conditions. Moreover, the array-oriented physical data organization proves to be extremely efficient (super linear) in the management of (very) long time series.

Table 4. Summary of results for the array-oriented test.

Array length	Execution time [s]	Throughput [GB/s]	Data size [GB]
12	1.8	0.6	1
120	2.1	4.9	10.3
1200	3.9	26.4	103
12000	18.9	54.5	1030

5 Related Work

Data analytics in eScience requires solutions able to manage and process large-scale data, taking into account several aspects, such as (i) the multi-dimensional nature of the datasets, (ii) the relevance of metadata for analysis purposes and (iii) the peculiarities of domain-specific algorithms.

The tools typically used for scientific data processing/analysis are client-side and operate mostly sequentially. Their inherent design does not make them particularly suited to target huge amounts of data. Indeed, such tools do not rely on distributed data storage and parallel processing, often failing for the lack of hardware resources (primarily RAM) on the execution node. To give some examples in the climate change domain, tools like CDO [22], NCO [26], ICCLIM[3], NCL[4] are successfully and largely adopted, very well-known, but unfortunately they are not designed to straightforwardly meet large data volume requirements and scenarios.

SciDB [8,24] and Rasdaman [4–6] are eScience-oriented projects aimed at addressing the above issues. SciDB is actually a distributed non-relational DBMS supporting full ACID properties and based on a multi-dimensional array oriented data model. Its set of statistical and linear algebra operations can be easily extended with user-defined types and user-defined functions. SciDB has been effectively used for data analysis in various scientific domains [23]. Rasdaman ("raster data manager") is an array database designed to store and query massive multi-dimensional arrays-based data, like images, simulation and sensors from various scientific domains. Similarly to Ophidia, they rely on n-dimensional arrays and offer a server side and declarative approach, although Ophidia is rather a framework centered around the datacube abstraction, providing an HPC-based environment for parallel data processing, metadata/provenance management and OLAP. Furthermore, Ophidia relies on a custom imperative-declarative approach to specify operations and implements interfaces for the creation and execution of high-level Python-based applications (through Ophidia python bindings, PyOphidia[5]) and scientific workflows.

[3] ICCLIM (Indice Calculation CLIMate) https://icclim.readthedocs.io/en/latest/intro.html.
[4] NCAR command language https://www.ncl.ucar.edu/.
[5] PyOphidia - Conda Forge https://anaconda.org/conda-forge/pyophidia.

Some efforts focusing on the extension of general purpose systems to support more scientific-oriented data analytics and formats have also recently emerged in literature. In particular, the well-known Spark parallel computing framework has been extended in various projects. SciSpark [20,25], for example, extends the Spark framework with the Scientific Resilient Distributed Dataset (sRDD), a distributed in-memory array structure for multi-dimensional data designed to support scientific data structures and algorithms. In particular, it targets the weather and climate change domains providing features to read data in parallel directly from HDF [16] and NetCDF file formats and a common interface to multiple linear algebra libraries. Another similar example is ClimateSpark [19], which also extends Spark to support climate change data analysis by defining an extension of RDD, called ClimateRDD, i.e., an immutable in-memory distributed collection of climate data chunks capable of managing multi-dimensional arrays. The system also provides some domain-specific transformations performed in parallel. These tools provide better support for scientific analysis, with respect to general purpose frameworks, while exploiting the power of the Spark programming and computing framework. However, differently from Ophidia, these solutions have not been originally designed for HPDA, using a parallel shared I/O and HPC-based computation model, or do not yet provide full support for domain-specific metadata management.

Another effort that is currently in the spotlight in the scientific community is Dask[6], a flexible library for parallel computing in Python. Eco-systems like Pangeo[7] are exploiting Dask for parallel climate data analysis. With respect to Dask, Ophidia includes (i) a very flexible and robust I/O layer (thus providing both compute- and storage-level capabilities), (ii) a more integrated approach with HPC-based parallel paradigms (i.e. MPI, OpenMP).

6 Conclusions

In the eScience landscape, the Ophidia project provides a High Performance Data Analytics framework joining HPC paradigms with scientific data analysis approaches to tackle large-scale parallel climate change data analysis. The core aspects of Ophidia, such as its storage model design and related distributed and hierarchical implementation across multiple physical storage resources are presented. Additionally, the experimental results (e.g., parallel efficiency and throughput) about a key analytical operator for data reduction, executed under different experimental setups (i.e., data partitioning and distribution conditions), are also discussed. Still, a specific benchmark of time series data reduction provides some insights in terms of scalability, efficiency and throughput of Ophidia. In particular, the results show how the Ophidia data distribution and partitioning enable the parallel data reduction operator to scale up to the full capacity of our cluster (more than 70% in all the cases analyzed). As a future work, a large-scale benchmark running on Marenostrum (PRACE Tier0 machine at

[6] Dask, library for dynamic task scheduling https://dask.org.
[7] Pangeo. A community platform for big data geoscience. https://pangeo.io/.

Barcelona Supercomputing Center) to evaluate the performance results of other classes of Ophidia operators (e.g., I/O), as well as a set of selected end-users applications, will be performed in the context of the ESiWACE Center of Excellence on Weather and Climate Simulations in Europe project[8]. Additionally, Ophidia will be further extended to support the Earth System Data Middleware[9] interface, developed in the ESiWACE project, to enable advanced scenarios at extreme-scale, like scalable in-situ visualization on HPC machines of global, high-resolution climate change simulation datasets.

Acknowledgments. This work was supported in part by the EU H2020 Excellence in SImulation of Weather and Climate in Europe (ESiWACE) project (Grant Agreement 675191). Moreover, the authors would like to acknowledge Antonio Aloisio for his editing and proofreading work on this paper.

References

1. Aloisio, G., Fiore, S.: Towards exascale distributed data management. Int. J. High Perform. Comput. Appl. **23**(4), 398–400 (2009). https://doi.org/10.1177/1094342009347702
2. Aloisio, G., Fiore, S., Foster, I., Williams, D.: Scientific big data analytics challenges at large scale. Proceedings of Big Data and Extreme-scale Computing (BDEC) (2013)
3. Asch, M., et al.: Big data and extreme-scale computing: pathways to convergence-toward a shaping strategy for a future software and data ecosystem for scientific inquiry. Int. J. High Perform. Comput. Appl. **32**(4), 435–479 (2018). https://doi.org/10.1177/1094342018778123
4. Baumann, P., Dehmel, A., Furtado, P., Ritsch, R., Widmann, N.: The multidimensional database system RasDaMan. SIGMOD Rec. **27**(2), 575–577 (1998). https://doi.org/10.1145/276305.276386
5. Baumann, P., Dehmel, A., Furtado, P., Ritsch, R., Widmann, N.: Spatio-temporal retrieval with RasDaMan. In: Proceedings of the 25th International Conference on Very Large Data Bases, VLDB 1999 pp. 746–749. Morgan Kaufmann Publishers Inc., San Francisco (1999). http://dl.acm.org/citation.cfm?id=645925.671513
6. Baumann, P., Furtado, P., Ritsch, R., Widmann, N.: The RasDaMan approach to multidimensional database management. In: Proceedings of the 1997 ACM Symposium on Applied Computing, SAC 1997, pp. 166–173. ACM, New York (1997). https://doi.org/10.1145/331697.331732
7. Bell, G., Hey, T., Szalay, A.: Beyond the data deluge. Science **323**(5919), 1297–1298 (2009). https://doi.org/10.1126/science.1170411
8. Brown, P.G.: Overview of sciDB: large scale array storage, processing and analysis. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, pp. 963–968. ACM, New York (2010). https://doi.org/10.1145/1807167.1807271

[8] The ESiWACE Center of Excellence on Weather and Climate Simulations in Europe project https://www.esiwace.eu/.

[9] ESiWACE Earth System Data Middleware https://github.com/ESiWACE/esdm.

9. D'Anca, A., et al.: On the use of in-memory analytics workflows to computer science indicators from large climate datasets. In: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 1035–1043, May 2017. https://doi.org/10.1109/CCGRID.2017.132
10. Dongarra, J., et al.: The international exascale software project roadmap. Int. J. High Perform. Comput. Appl. **25**(1), 3–60 (2011). https://doi.org/10.1177/1094342010391989
11. Elia, D., et al.: An in-memory based framework for scientific data analytics. In: Proceedings of the ACM International Conference on Computing Frontiers, CF 2016, pp. 424–429. ACM, New York (2016). https://doi.org/10.1145/2903150.2911719
12. Fiore, S., et al.: Ophidia: a full software stack for scientific data analytics. In: 2014 International Conference on High Performance Computing Simulation (HPCS), pp. 343–350, July 2014. https://doi.org/10.1109/HPCSim.2014.6903706
13. Fiore, S., et al.: Distributed and cloud-based multi-model analytics experiments on large volumes of climate change data in the earth system grid federation ecosystem. In: 2016 IEEE International Conference on Big Data (Big Data), pp. 2911–2918, December 2016. https://doi.org/10.1109/BigData.2016.7840941
14. Fiore, S., D'Anca, A., Palazzo, C., Foster, I.T., Williams, D.N., Aloisio, G.: 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 (2013). https://doi.org/10.1016/j.procs.2013.05.409
15. Fiore, S., et al.: Big data analytics on large-scale scientific datasets in the INDIGO-DataCloud project. In: Proceedings of the Computing Frontiers Conference, CF 2017, pp. 343–348. ACM, New York (2017). https://doi.org/10.1145/3075564.3078884
16. Folk, M., Heber, G., Koziol, Q., Pourmal, E., Robinson, D.: An overview of the HDF5 technology suite and its applications. In: Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases. AD 2011, pp. 36–47. ACM, New York (2011). https://doi.org/10.1145/1966895.1966900
17. Golfarelli, M., Rizzi, S.: Data Warehouse Design: Modern Principles and Methodologies, 1st edn. McGraw-Hill Inc., New York (2009)
18. Gray, J., Liu, D.T., Nieto-Santisteban, M., Szalay, A., DeWitt, D.J., Heber, G.: Scientific data management in the coming decade. SIGMOD Rec. **34**(4), 34–41 (2005). https://doi.org/10.1145/1107499.1107503
19. Hu, F., et al.: ClimateSpark: an in-memory distributed computing framework for big climate data analytics. Comput. Geosci. **115**, 154–166 (2018). https://doi.org/10.1016/j.cageo.2018.03.011
20. Palamuttam, R., et al.: SciSpark: applying in-memory distributed computing to weather event detection and tracking. In: 2015 IEEE International Conference on Big Data (Big Data), pp. 2020–2026, October 2015. https://doi.org/10.1109/BigData.2015.7363983
21. Reed, D.A., Dongarra, J.: Exascale computing and big data. Commun. ACM **58**(7), 56–68 (2015). https://doi.org/10.1145/2699414
22. Schulzweida, U.: CDO user guide - version 1.9.6 (2019). https://code.mpimet.mpg.de/projects/cdo/embedded/cdo.pdf
23. Stonebraker, M., Brown, P., Becla, J., Zhang, D.: SciDB: a database management system for applications with complex analytics. Comput. Sci. Eng. **15**(3), 54–62 (2013). https://doi.org/10.1109/MCSE.2013.19

24. Stonebraker, M., Brown, P., Poliakov, A., Raman, S.: The Architecture of SciDB. In: Bayard Cushing, J., French, J., Bowers, S. (eds.) SSDBM 2011. LNCS, vol. 6809, pp. 1–16. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22351-8_1
25. Wilson, B., et al.: SciSpark: highlyinteractive in-memory science data analytics. In: 2016 IEEE InternationalConference on Big Data (Big Data), pp. 2964–2973, December 2016. https://doi.org/10.1109/BigData.2016.7840948
26. Zender, C.S.: Analysis of self-describing gridded geoscience data with netCDF Operators (NCO). Environ. Model. Softw. **23**(10), 1338–1342 (2008). https://doi.org/10.1016/j.envsoft.2008.03.004