Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2013) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

2013 International Conference on Computational Science

Ophidia: toward big data analytics for eScience

S. Fiore[a,b]*, A. D'Anca[a], C. Palazzo[a,b], I. Foster[c], D. N. Williams[d], G. Aloisio[a,b]

[a]*Euro-Mediterranean Center on Climate Change, Italy*
[b]*University of Salento, Italy*
[c]*University of Chicago and Argonne National Laboratory, US*
[d]*Lawrence Livermore National Laboratory, US*

Abstract

This work introduces Ophidia, a big data analytics research effort aiming at supporting the access, analysis and mining of scientific (*n*-dimensional array based) data. The Ophidia platform extends, in terms of both primitives and data types, current relational database system implementations (in particular MySQL) to enable efficient data analysis tasks on scientific array-based data. To enable big data analytics it exploits well-known scientific numerical libraries, a distributed and hierarchical storage model and a parallel software framework based on the Message Passing Interface to run from single tasks to more complex dataflows. The current version of the Ophidia platform is being tested on NetCDF data produced by CMCC climate scientists in the context of the international Coupled Model Intercomparison Project Phase 5 (CMIP5).

Keywords: data warehouse; OLAP framework; parallel computing; scientific data management; big data.

1. Introduction

Data analysis and mining on large data volumes have become key tasks in many scientific domains [1]. Often, such data (e.g., in life sciences [2], climate [3], astrophysics [4], engineering) are multidimensional and require specific primitives for subsetting (e.g., slicing and dicing), data reduction (e.g., by aggregation), pivoting, statistical analysis, and so forth. Large volumes of scientific data strongly need the same kind of On-Line Analytical Processing (OLAP) primitives typically used to carry out data analysis and mining [5]. However, current general-purpose OLAP systems are not adequate in big-data scientific contexts for several reasons:

* Corresponding author. Tel.: +39 0832 297371; fax: +39 0832 297371.
E-mail address: sandro.fiore@unisalento.it

- they do not scale up with the current size of scientific data (terabytes or petabytes). For example, a single climate simulation can produce tens of terabytes—and hundreds in the near future—of output [6]. A set of experiments would easily reach the exabyte scale in the next years;
- they do not provide a specific *n*-dimensional *array-based* data type (key for such kind of data);
- they do not provide *n*-dimensional *array-based* domain-specific functions (e.g. re-gridding for geospatial maps or time series statistical analysis).

In several disciplines, scientific data analysis on multidimensional data is made possible by using domain-specific tools, libraries, and command line interfaces that provide the needed analytics primitives. However, these tools often fail at the tera- to petabyte scale because (i) they are not available in parallel versions, (ii) they do not rely on scalable storage models (e.g., exploiting partitioning, distribution and replication of data) to deal with large volumes of data, (iii) they do not provide a declarative language for complex dataflow submission, and/or (iv) they do not expose a server interface for remote processing (usually they run on desktop machines through command line interfaces and need, as a preliminary step, the download of the entire raw data).

This work provides a complete overview of the Ophidia project, a big data analytics research effort aiming at facing the four cited challenges. In particular, Ophidia extends, in terms of both Structured Query Language (SQL) primitives and data types, current relational database systems (in particular MySQL) to enable efficient data analysis tasks on scientific array-based data. It exploits a proprietary storage model (exploiting data partitioning, distribution, and replication) jointly with a parallel software framework based on the Message Passing Interface (MPI) to run from single tasks to more complex dataflows. The SQL extensions work on *n*-dimensional arrays, rely on well-known scientific numerical libraries, and can be composed to compute multiple tasks into a single SQL statement. Further, Ophidia provides a server interface that makes the data analysis task a server-side activity in the scientific chain. Exploiting such an approach, most scientists would not need to download large volumes of data for their analysis as it happens today. On the contrary they would download the results of their computations (typically in the megabytes or even kilobytes order, like a jpg related to a chart or a map) after running multiple remote data analysis tasks. Such an approach could strongly change the daily activity of scientists by reducing the amount of data transferred on the wire, the time needed to carry out analysis tasks, and the number and the heterogeneity of the analysis software installed on client machines, leading to increased scientific productivity.

The remainder of this work is organized as follows. Section 2 presents some related works in the same area. Section 3 briefly describes main challenges and requirements. Section 4 presents the Ophidia implementation of the multidimensional data model. It describes the path from the use cases to the architectural design to the infrastructural implementation. Section 5 presents the Ophidia architecture. Section 6 discusses in particular the provided array-based primitives and describes practical examples in the climate change domain. Section 7 presents the conclusions and future work.

2. Related Work

The Ophidia project falls in the big data analytics area applied to eScience (it *is about global collaboration in key areas of science, and the next generation of infrastructure that will enable it* [7]) contexts. Other research projects are addressing similar issues with different approaches concerning the adoption of server-side processing, parallel implementations, declarative languages, new storage models, and so forth. Some of these other projects can be considered *general purpose* and some *domain specific*.

Two related projects in the first class are SciDB [8] and Rasdaman [9]. Both share with Ophidia a special regard for the management of *n*-dimensional arrays, server-side analysis, and declarative approach. However, the three systems provide different design and implementation with regard to the physical design, storage structure, query language statements, and query engine. Unlike the other two systems, Ophidia provides full support for both shared-disk and shared-nothing architectures jointly with the hierarchical data distribution scheme. On the other hand, Ophidia does not support versioning as in SciDB, nor does it provide OGC-compliant interfaces as in Rasdaman.

Concerning the second class of related work, for each scientific domain several research projects provide utilities for analyzing and mining scientific data. A cross-domain evaluation of this software reveals that most have the same issues and drawbacks, although they apply to different contexts. The following example will give an idea.

In the climate change domain, several libraries and command line interfaces are available, such as the Climate Data Operators (CDO) [10], NetCDF Operators (NCO) [11], Grid Analysis and Display System (GrADS) [12], and NCAR Command Language (NCL) [13]. None exploits a client-server paradigm jointly with a parallel implementation of the most common and used data operators, as does Ophidia. Moreover, a declarative and extensible approach is missing for running complex queries against big data archives; this approach is a strong requirement addressed by the Ophidia platform. So far, most efforts in this area have been devoted to develop scripts, libraries, command line interfaces, and visualization and analysis tools for desktop machines. Such an approach is not adequate with the current size of data and will definitely fail in the near petascale to exascale future when the output of a single climate simulation will be on the order of terabytes. The same can be stated in many other scientific domains, including geosciences, astrophysics, engineering, and life sciences.

3. Main challenges, needs and requirements

The Ophidia framework has been designed to address scalability, efficiency, interoperability, and modularity requirements. *Scalability* (in terms of management of large data volumes) needs a different way and approach to organize the data on the storage system. It involves new storage models, data partitioning, and data distribution scenarios (see Sections 4.2 and 4.3). Replication also addresses high-throughput use cases, fault tolerance, and load balancing and can be key in petascale and exascale scenarios to efficiently address resilience [5]. *Efficiency* affects parallel I/O and parallel solutions as well as algorithms at multiple levels (see Section 4.4). *Interoperability* can be achieved by adopting well-known and largely adopted standards at different levels (server interface, query language, data formats, communication protocols, etc.). *Modularity* is strongly connected to the system design and is critical for large frameworks such as Ophidia.

In terms of challenges, a *client-server paradigm* (instead of a desktop-based approach) is relevant to allow server-side computations. In this regard, a *parallel framework* can enable and address efficient and scalable data analysis (see Section 4.4), and a *declarative language* can provide the proper support for the management of scientific dataflow-oriented computations. Additional requirements coming from different domains are more *functional* and concern with *n*-dimensional array manipulation. Some of them are *data reduction operators*, *statistical analysis*, *subsetting*, *data filtering,* and *data transformation*.

4. Multidimensional data model: the Ophidia implementation

In the following subsections, starting from the multidimensional data model, the classic star schema implementation and the one proposed in the Ophidia project are presented and compared, highlighting the main differences regarding the related storage models.

4.1. Multidimensional data model and star schema

Scientific data are often multidimensional. A multidimensional data model is typically organized around a central theme (it is *subject oriented*) and views the data in the form of a *data cube*. It consists of several dimensions and measures. The measures are 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 is an evolution of the star schema. By definition, in this schema, the data warehouse implementation consists of a large central table (the *fact table*, FACT) that contains all the data with no redundancy and a set of smaller tables (called *dimension tables*), one for each dimension. The dimensions can also implement concept hierarchies, which provide a way for analyzing and mining at multiple levels of abstraction over the same dimension [14].

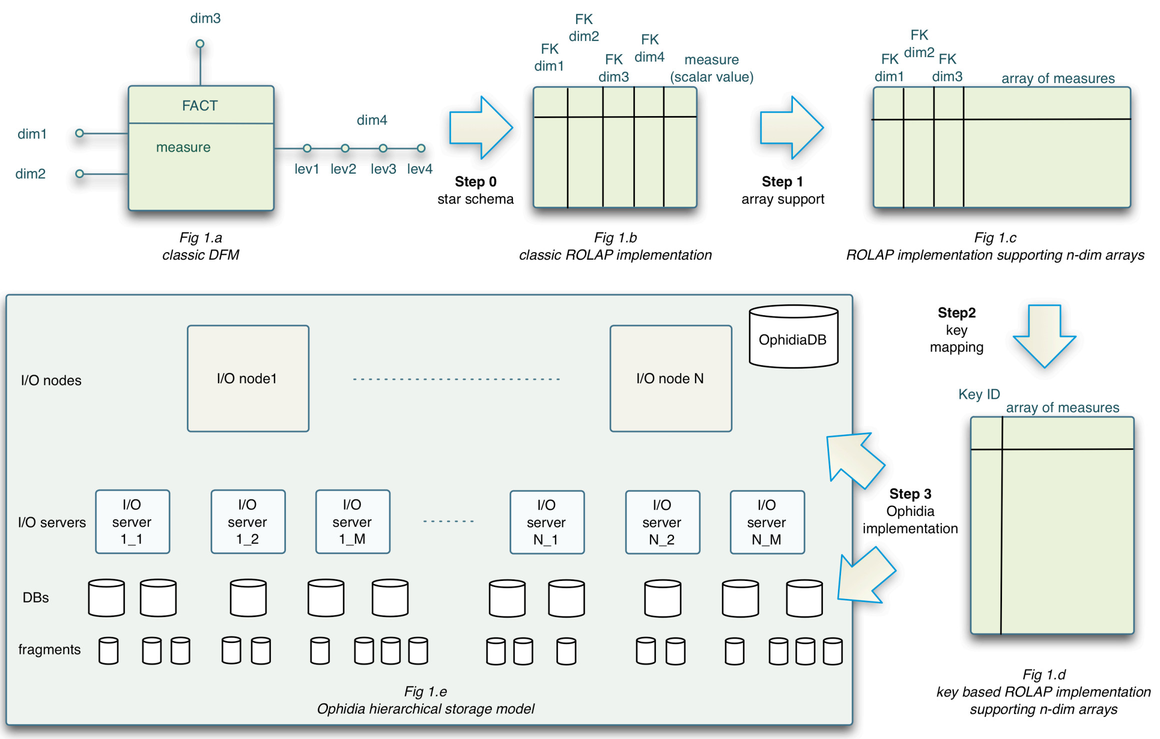


Fig 1. Moving from the DFM (Fig. 1.a) to the Ophidia hierarchical storage model (Fig.1.e)

Let us consider the Dimensional Fact Model [15] (DFM, a conceptual model for data warehouse) depicted in Fig. 1.a. The classic Relational-OLAP (ROLAP) based implementation of the associated star schema is presented in Fig. 1.b. There is one fact table (FACT); four dimensions (dim1, dim2, dim3, and dim4), with the last dimension modeled through a 4-level concept hierarchy (lev1, lev2, lev3, lev4); and a single measure (measure). To better understand the Ophidia internal storage model, we can consider a NetCDF output of a global model simulation where dim1, dim2, and dim3 correspond to *latitude*, *longitude*, and *depth*, respectively; dim4 is *time*, with the concept hierarchy *year*, *quarter*, *month*, *day*; and measure is *air pressure*.

4.2. Ophidia internal storage model

The Ophidia internal storage model is a two-step-based evolution of the star schema. The first step includes the support for array-based data types (see Fig 1.c), and the second step includes a key mapping related to a set of foreign keys (fks) (see Fig 1.d). These two steps allow a multidimensional array to be managed on a single tuple (e.g., an entire time series on the same row) and the n-tuple (fk_dim1, fk_dim2, …, fk_dim*n*) to be replaced by a single *key* (a numerical *ID*). The second step makes the Ophidia storage model and implementation independent of the number of dimensions, unlike the classic ROLAP-based implementation. The system moves from a ROLAP approach to a relational key-array approach supporting *n*-dimensional data management and a reduced disk space devoted to store both the arrays and the table indexes. The *key* attribute manages with a single *ID* a set of *m* dimensions ($m<n$, mapped onto the *ID* through a numerical function *ID* = *f*(fk_dim1, fk_dim2,…, fk_dim*m*)) called *explicit* dimensions. The *array* attribute manages the other *n-m* dimensions, called *implicit* dimensions.

In our example, latitude, longitude and depth are explicit dimensions, while time is implicit (1-D array). The mapping on the Ophidia key-array data storage model consists of having 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 RDBMS 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 several array-based primitives to manage arrays stored through the Ophidia storage model. We provide additional details about a subset of these primitives, their interfaces (API) and some examples in the following sections.

4.3. Hierarchical data management

We must address several scalability issues in order to manage large volumes of data. Several optimizations in the physical design of the system have to be implemented in order to improve the overall efficiency (see Fig. 1.e). In particular, data partitioning and distribution can be important in scientific domains where the users have to deal with a huge amount of data. In the following, we discuss the horizontal partitioning technique that we use jointly with a hierarchical storage structure. Horizontal partitioning does not require additional disk space but does add complexity at the software level. It consists of splitting the central FACT table by *ID* into multiple smaller tables (chunks called *fragments*). Many queries can execute more efficiently when using horizontal partitioning since it allows parallel query implementations and only a small fraction of the fragments may be involved in query execution (e.g., subsetting task). The fragments produced by the horizontal partitioning scheme are mapped onto a hierarchical structure composed of four different levels:

- *Level 0*: multiple I/O nodes (multi-host);
- *Level 1*: multiple instances of DBMS on the same I/O node (multi-DBMS);
- *Level 2*: multiple instances of physical databases on the same DBMS (multi-DB);
- *Level 3*: multiple fragments on the same physical database (multi-table).

The values related to these levels are key settings for the fragments distributions. Fine tuning these parameters will be part of the future work, as the performance of the system can truly depend on them.

4.4. Basic foundations about the Ophidia parallel framework

Because of the hierarchical data storage organization, the Ophidia platform has to provide a comprehensive set of parallel operators to carry out data analysis and mining on a large set of distributed fragments as a whole and, more generally, to manipulate data cubes. We describe here only the basic foundations of the parallel framework; more details will be provided in a future work, starting from the algebra connected with this approach. We plan for three classes of parallel operators in order to compute distributive measures, that *can be computed for a given data set by partitioning the data into smaller subsets, computing the operator for each* *subset, and then merging the results in order to arrive at the measure's value for the original (entire) data set*; algebraic ones, that *can be computed by applying an algebraic function to one or more distributive measures*; and holistic ones, that *must be computed on the entire data set as a whole* [14]. To date, only a subset of the distributive operators has been implemented. We plan to implement algebraic operators next, and then holistic operators by the application of specific (distributive or algebraic) heuristics.

The resulting parallel framework is not complex, and its algorithm relies on a few steps:

- distribute data (input fragments) among available processes/threads;
- apply the distributive operator in parallel on each fragment (for instance, data reduction, by aggregating on the implicit dimension of the array, which in our use case could be from daily to monthly data);
- store the output of the distributive operator as a new set of fragments.

The current version of the Ophidia framework can perform on the entire data cube (and in parallel) the following (sequential) fragment-level tasks:

- subsetting (slicing and dicing, both along the explicit and the implicit dimensions);
- data reduction (by aggregating in terms of sum, maximum, minimum, count, both on the implicit and on the explicit dimensions);
- predicate evaluation on the array data;
- basic array-oriented primitives, like *sum a scalar to the array* or *multiply an array by a scalar*, *sum two arrays*, and *concatenate two arrays*;
- a composition of the aforementioned tasks.

The parallel command line interface supplies the user with additional I/O functionalities such as import (export) from (to) specific formats like NetCDF, Grib, HDF, and CSV to (from) the Ophidia storage infrastructure. Additional details about the fragment-level primitives (which have a bigger focus in this paper) are reported in Section 6. The Ophidia system is currently being tested at CMCC on CMIP5 [16] data in NetCDF [17] format Climate and Forecast (CF) convention compliant (about 100TB of data published in the Earth System Grid Federation) by a beta-test group of climate scientists. Preliminary tests on a prototype environment demonstrate that the system scales well both in a shared-nothing and shared-disk (exploiting GPFS) configuration. A detailed benchmark and a comprehensive performance evaluation related to all the Ophidia operators cannot be reported in this paper because of page limit issues and will be published in a future work, once all the tests are completed and the experimental results validated.

5. The Ophidia architecture

This section presents the Ophidia architecture, motivating specific choices that have been made to address the system requirements mentioned in Section 3.

The Ophidia architecture consists of (i) the server front-end, (ii) the OphidiaDB, (iii) the compute nodes, (iv) the I/O nodes and (v) the storage system. We describe each component in detail.

The *server front-end* is responsible for accepting and dispatching incoming requests. It is a prethreaded server implementing a standard WS-I interface. We rely on X.509 digital certificates for authentication and Access Control List (ACL) for authorization. The *OphidiaDB* is the system (relational) database. By default the server front-end uses a MySQL database to store information about the system configuration and its status, available data sources, registered users, available I/O servers, and the data distribution and partitioning. The *compute nodes* are computational machines used by the Ophidia software to run the parallel data analysis operators. The *I/O nodes* are the machines devoted to the parallel I/O interface to the storage. Each I/O node hosts one or more I/O servers responsible for I/O with the underlying storage system described in Section 4. The I/O servers are MySQL DBMSs supporting, at both the data type and primitives levels, the management of *n*-dimensional array structures. This support has been integrated into the classic MySQL software by mapping the array structure on the MySQL binary data type and adding a new set of functions (exploiting the User

Defined Function approach, UDF) to manipulate arrays (see next section). The *storage system* is the hardware resource managing the data store, that is, the physical resources hosting the data according to the hierarchical storage structure defined in Section 4.

6. Array-based primitives in Ophidia

An RDBMS usually supports basic data types such as integer, float, double, string (which also includes binary), time and date. Additionally, several DBMS implementations support spatial data types to represent geometry objects; examples are the MySQL spatial extensions or *PostGIS, a Postgresql extension* with support for geographic objects. However, the array data type is not usually fully supported by an RDBMS as a native data type and with a complete set of manipulation primitives; in some cases, the array data type is supported, but the available methods do not meet the needs of scientific data analysis.

A UDF offers a powerful way to extend the functionality of a MySQL database, by adding new functions that are not available (by default) in the system; other ways are by means of built-in functions compiled in the *mysqld server* or through *stored functions*. Ophidia uses this mechanism to provide both the array data type (mapped onto the MySQL binary data type) and a wide set of primitives to manipulate it. Tables 1, 2, and 3 list the functions included in two classes of plugins: tuple-level and table-level. Tuple-level plugins work only at the array level. These plugins include both core (e.g., max of an array) and more complex functions (e.g., five-number summary of an array: median, quartiles Q1 and Q3, and smallest and largest values). In both cases, the Ophidia plugins exploit well-known and widely adopted scientific libraries, such as the *GNU Scientific Library* (GSL) [18] and the *Portable, Extensible Toolkit for Scientific Computation* (PETSc) [19]. The single table-level plugin operates on a set of arrays to aggregate data via max, min, count, sum and average operators.

We note the following:

- most plugins take an array as input and provide an array as output; this property allows a very *simple* composition (nesting) of plugins to perform more semantically *complex* tasks (query);
- all plugins (currently available as sequential implementations) represent extensions of the SQL, so that they can be embedded in SQL statements to perform scientific data analysis tasks;
- the argument (whose data type is denoted as *byte**) refers to the binary array (input argument), that is the MySQL table attribute storing the array of data;
- optional arguments are enclosed in square brackets; in particular *oph_type* (that can be set to *OPH_INT*, *OPH_FLOAT*, *OPH_DOUBLE*) and *oph_operator* (that can be set to *OPH_MAX*, *OPH_ MIN*, *OPH_SUM*, *OPH_AVG*, *OPH_STDEV*) respectively refer to the data type of the single element of the array and the operator that will be applied to the array (e.g., *oph_reduce*).

We use a few examples to illustrate some of the functionalities currently implemented.

Table 1. Tuple level plugins – core functions

Primitives API	Primitives description
long *oph_count_array* (byte* measure, [oph_type t])	Return the number of elements in the array
byte* *oph_subarray* (byte* measure, [long long start], [long long count], [oph_type t])	Extract a sub-array of *count* elements starting from the *start* element
byte* *oph_sum_scalar* (byte* measure, [double value], [oph_type t])	Add a scalar *value* to each element of the array (the data type of *value* is automatically converted)
byte* *oph_mul_scalar* (byte* measure, [double value], [oph_type t])	Multiply by a scalar *value* each element of the array (the data type of *value* is automatically converted)
byte* *oph_reduce* (byte* measure, [oph_operator op], [long long count], [oph_type t])	Aggregate the elements of the array by applying a specific operator *op* on sets of *count* elements
byte* *oph_sum_array* (byte* measure a, byte* measure b, [oph_type ta], [oph_type tb])	Sum two arrays

long *oph_find* (byte* measure, double value, [long long distance], [oph_type t])	Return the number of elements having a distance from *value* less than the *distance* argument
byte* *oph_shift* (byte* measure, [long offset], [double filling], [oph_type t])	Shift the elements of an array by *offset* positions by adding a 0-padding (or a *filling*-padding)
byte* *oph_rotate* (byte* measure, [long offset], [oph_type t])	Rotate the elements of an array by *offset* positions
byte* *oph_reverse* (byte* measure, [oph_type t])	Reverse the elements in the array
byte* *oph_concat* (byte* measure a, byte* measure b, [oph_type ta], [oph_type tb])	Join the arrays *a* and *b*

byte* *oph_predicate* (byte* measure, char* expression0, char* comparison, char* expression1, char* espression2, [oph_type t])	FOR *i*=0 TO *oph_count_array*(*measure*)
	  IF (*expression0(measure[i])* IS *'comparison'*)
	    THEN return *expression1*(*measure*[*i*]))
	  ELSE return *expression2*(*measure*[*i*]))
	
	- *comparison* can be >0, <0, >=0, <=0, ==0, !=0
	- *expression0*(*x*), *expression1*(*x*) and *expression2*(*x*) are three mathematical one-variable functions

char* *oph_dump* (byte* measure, [oph_type t])	Convert the array from binary to a human-readable set of elements

Table 2. Tuple level plugins – functions based on the GSL scientific library

Primitives API	Primitives description
byte* *oph_gsl_sd* (byte* measure, [oph_type t])	Evaluate the standard deviation of an array exploiting the GSL library
byte* *oph_gsl_boxplot* (byte* measure, [oph_type t])	Evaluate the five-number summary of an array exploiting the GSL library

Table 3. Table level plugins

Primitive API	Primitive description
byte* *oph_aggregate_operator* (byte* measure, [oph_operator op], [oph_type t])	It carries out a vertical aggregation by applying the *op* operator across several tuples

A few examples can be useful to better understand how *n*-dimensional arrays of data can be easily analyzed. The following example relates to the *oph_gsl_boxplot* plugin. The associated SQL statement is:

```
SELECT oph_dump(oph_gsl_boxplot(measure)) FROM fact;
```

For each array the output of the plugin is a five-number summary (itself an array). Figures 2 and 3 show the transformation performed by this plugin from both a storage level and a scientific point of view. This plugin has been designed to meet the requirements coming from a climate change use case (defined with CMCC scientists) related to the analysis of extreme events. The use case analyzes the output of global climate simulations and, in particular, the maximum daily temperature in terms of probability density function over land in the 1966–2005 timeframe. Another domain-based example in the climate change context involves a monthly-based data reduction carried out on a subset (four months, from January to April) of daily data (360-value based arrays according to a 360-day calendar, 12 months of 30 days each). The related SQL statement is:

```
SELECT oph_dump(oph_reduce(oph_subarray(measure,1,120),'OPH_AVG',30)) FROM fact;
```

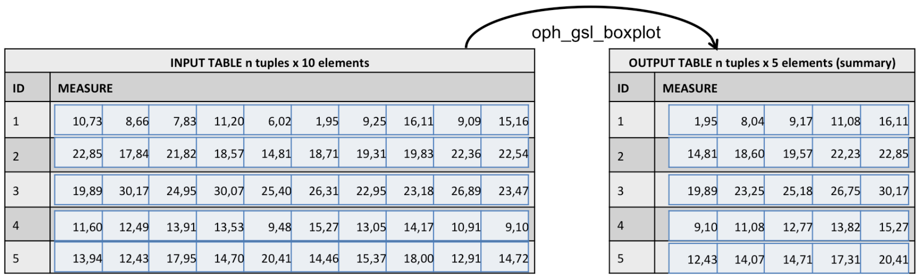
which requests a monthly-based data reduction on the 120-value arrays got from the *oph_subarray* primitive applied on the first four months, or as in the following:

```
SELECT oph_dump(oph_subarray(oph_reduce(measure,'OPH_AVG',30),1,4)) FROM fact;
```

which means subsetting the first four values in the monthly-based arrays obtained from the *oph_reduce* primitive. Let us consider now the following SQL statement:

```
SELECT oph_dump(oph_predicate(measure,'x-10','>0','0','1')) from fact;
```

This example applies the *oph_predicate* plugin to the measure array by computing for each element of the array the following expression: measure[*i*]-10. If the result is greater than zero, the element at index *i* in the output array is set to 1; otherwise it is set to 0.


oph_gsl_boxplot

INPUT TABLE n tuples x 10 elements

ID	MEASURE									
1	10,73	8,66	7,83	11,20	6,02	1,95	9,25	16,11	9,09	15,16
2	22,85	17,84	21,82	18,57	14,81	18,71	19,31	19,83	22,36	22,54
3	19,89	30,17	24,95	30,07	25,40	26,31	22,95	23,18	26,89	23,47
4	11,60	12,49	13,91	13,53	9,48	15,27	13,05	14,17	10,91	9,10
5	13,94	12,43	17,95	14,70	20,41	14,46	15,37	18,00	12,91	14,72

OUTPUT TABLE n tuples x 5 elements (summary)

ID	MEASURE				
1	1,95	8,04	9,17	11,08	16,11
2	14,81	18,60	19,57	22,23	22,85
3	19,89	23,25	25,18	26,75	30,17
4	9,10	11,08	12,77	13,82	15,27
5	12,43	14,07	14,71	17,31	20,41

Fig. 2. Input (left) and output (right) of the *oph_gsl_boxplot* operator.

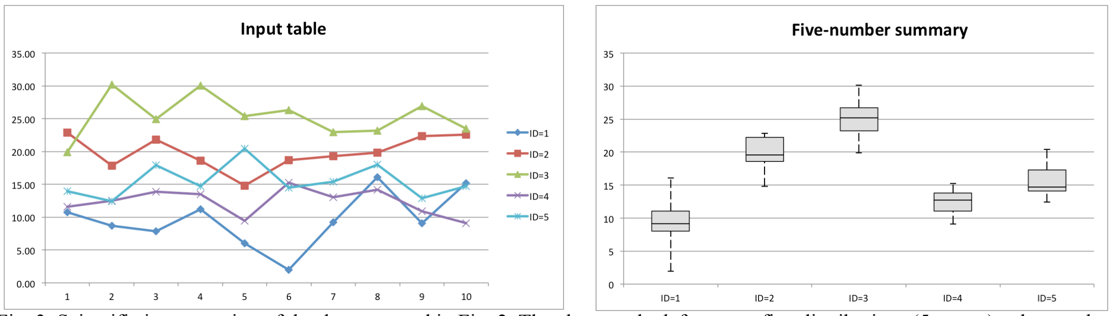


Fig. 3. Scientific interpretation of the data reported in Fig. 2. The chart on the left reports five distributions (5 arrays), whereas the boxplot on the right (which is the output of the *oph_gsl_boxplot* plugin) reports the associated five-number summary.

The example could become more complex if, as in the following query, we request a subarray of measure (e.g., just 5 elements from index 3 to 7) and define two mathematical expressions (such as *x*x* and *2*x*) instead of the constants 0 and 1:

```
SELECT oph_dump(oph_predicate(oph_subarray(measure,3,5),'x-10','>0','x*x','2*x' )) FROM fact;
```

This SQL statement shows clearly how several primitives (plugins) can be easily composed, like a mathematical expression, to perform complex operations on data by running a single query. Additional examples in the climate change domain, such as time series extraction, data aggregation, and data model inter-comparison, can be easily inferred by composing the array-based primitives provided in Table 1.

7. Conclusions and future work

We have presented Ophidia, a big data analytics research effort aiming at supporting access, analysis, and mining of *n*-dimensional array-based scientific data. Special emphasis has been given to the Ophidia internal storage model, the general architecture and the array-based primitives as they represent key pillars for this framework. Some details about the parallel framework and the plugins APIs also have been provided. We also remarked how this project joins basic foundations concerning data warehouse systems, new storage models, OLAP frameworks, numerical libraries and parallel paradigms in order to face big data analytics challenges in eScience domains. A set of use cases related to the climate change domain also has been provided to highlight the capabilities of the system as well as its declarative approach. Future work is related to the development of an extended set of parallel operators, to support both distributive and algebraic functions. Multiple interfaces are also part of the future activities (cloud-based, OGC-WPS [20] compliant, and grid-enabled are just the most relevant ones). Specific SQL extensions and an optimized query planner also will be considered in order to support more complex operators and dataflow driven requests.

Acknowledgements

This work was supported in part by the Italian Ministry of Education, Universities and Research (MIUR) under contract GEMINA and by the U.S. Department of Energy, Office of Science, under Contract DE-AC02-06CH11357.

References

[1] S. Fiore and G. Aloisio, Special section: Data management for eScience. *Future Generation Computer System* 27(3): 290-291 (2011).
[2] Julio Saez-Rodriguez, et al., Flexible informatics for linking experimental data to mathematical models via DataRail. *Bioinformatics* 24, 6 (March 2008), 840-847, 2008, doi-10.1093/bioinformatics/btn018 http://dx.doi.org/10.1093/bioinformatics/btn018.
[3] William Hendrix et al., Community Dynamics and Analysis of Decadal Trends in Climate Data, (ICDM Climate 2011).
[4] Rob Latham, et al., A case study for scientific I/O: improving the FLASH astrophysics code. Comput. Sci. Disc. 5 (2012) 015001.
[5] J. Dongarra, P. Beckman, et al., The International Exascale Software Project roadmap. International *J. High Performance Computing Apps*. 25, no. 1, 3-60 (2011), ISSN 1094-3420 doi: 10.1177/1094342010391989.
[6] G. Aloisio and S. Fiore, Towards exascale distributed data management, International *J. of High Performance Computing Apps*. 23, no. 4, 398-400 (2009) doi: 10.1177/1094342009347702.
[7] J. Taylor, Defining eScience http://www.nesc.ac.uk/nesc/define.html.
[8] M. Stonebraker, et al., The architecture of SciDB. In Proceedings of the 23rd international conference on Scientific and statistical database management, edited by J. B. Cushing, J. French, and S. Bowers (Eds.). Springer-Verlag, Berlin, Heidelberg, 2011, pp. 1-16.
[9] P. Baumann, et al., The multidimensional database system RasDaMan. SIGMOD Rec. 27, no. 2, 575-577 (June 1998). doi:10.1145/276305.276386 http://doi.acm.org/10.1145/276305.276386.
[10] Climate Data Operators (CDO) - https://code.zmaw.de/projects/cdo.
[11] C. S. Zender, Analysis of self-describing gridded geoscience data with netCDF Operators (NCO), Environmental Modelling & Software, 23, no. 10–11, pp. 1338–1342, 2008.
[12] P. Tsai and B.E. Doty, A Prototype Java interface for the Grid Analysis and Display System (GrADS). In Fourteenth International Conference on Interactive Information and Processing Systems, Phoenix, Arizona, 1998.
[13] The NCAR Command Language (Version 6.0.0) [Software]. (2012). Boulder, Colorado: UCAR/NCAR/CISL/VETS. http://dx.doi.org/10.5065/D6WD3XH5.
[14] J. Han and M. Kamber, *Data mining: Concepts and Techniques*, Morgan Kaufmann Publishers, 2005.
[15] M. Golfarelli. The DFM: A conceptual model for data warehouse. In Encyclopedia of Data Warehousing and Mining (2nd Edition), John Wang (Ed.), IGI Global, pp. 638-645, 2008.
[16] K.E. Taylor, R. J. Stouffer, and G. A. Meehl 2012 An overview of CMIP5 and the experiment design, Bulletin of the American Meteorological Society 93, no. 4, 485-498 (2012), doi:10.1175/BAMS-D-11-00094.1, http://journals.ametsoc.org/doi/abs/10.1175/BAMS-D-11-00094.1.
[17] R. K. Rew and G. P. Davis, The Unidata netCDF: Software for scientific data access. In 6th Int. Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology, *American Meteorology Society*, pp. 33-40, February 1990.
[18] The GNU Scientific Library (GSL) http://www.gnu.org/software/gsl/.
[19] Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, et al., PETSc web page http://www.mcs.anl.gov/petsc, 2012.
[20] OGC Web Processing Service, http://www.opengeospatial.org/standards/wps.