SAVIME: An Array DBMS for Simulation Analysis and ML Models Predictions

Hermano Lourenço Souza Lustosa, Anderson Chaves da Silva, Daniel Nascimento Ramos da Silva, Patrick Valduriez, Fábio André Machado Porto

SAVIME: An Array DBMS for Simulation Analysis and ML Models Predictions

Hermano L. S. Lustosa[1], Anderson C. Silva[1], Daniel N. R. da Silva[1], Patrick Valduriez[2], and Fabio Porto[1]

[1]National Laboratory for Scientific Computing, Rio de Janeiro, Brazil
[2]Inria, University of Montpellier, CNRS, LIRMM, France
[3]{hlustosa, achaves, dramos, fporto}@lncc.br,patrick.valduriez@inria.fr

February 15, 2021

Abstract

Limitations in current DBMSs prevent their wide adoption in scientific applications. In order to make them benefit from DBMS support, enabling declarative data analysis and visualization over scientific data, we present an in-memory array DBMS called SAVIME. In this work we describe the system SAVIME, along with its data model. Our preliminary evaluation show how SAVIME, by using a simple storage definition language (SDL) can outperform the state-of-the-art array database system, SciDB, during the process of data ingestion. We also show that it is possible to use SAVIME as a storage alternative for a numerical solver without affecting its scalability, making it useful for modern ML based applications.

Keywords: Scientific Data Management, Multidimensional Array, Machine Learning
[1]

1 Introduction

Due to the increasing computational power of HPC environments, vast amounts of data are now generated in different research fields, and a relevant part of this data is best represented as array data. Geospatial and temporal data in climate modeling, astronomical images, medical imagery, multimedia and simulation

[1]This work has been done in the context of the HPDaSc Inria Associated Team with Brazil and the internal partnership between Inria and LNCC. We would like to thank CAPES and CNPq for scholarships and research productivity fellowships, and Petrobras for financing this work through the Gypscie project.

data are naturally represented as multidimensional arrays. To analyze such data, DBMSs offer many advantages, like query languages, which eases data analysis and avoids the need for extensive coding/scripting, and a logical data view that isolates data from the applications that consume it.

The efficient execution of ad-hoc heavy-weight analytical queries is still an open problem, especially when efficiency means as fast as possible [18]. It is reasonable to expect that finding the best model to represent the data structures one is working with may lead to better results than trying to build a single general-purpose data model fit to every problem. For instance, benchmark experiments have shown a performance improvement of orders of magnitude when using array data models to analyze scientific data in a DBMS compared to more traditional models [37]. Furthermore, an appropriate DBMS data model provides a semantically richer data representation, which simplifies the creation of analytical queries and visualizations.

Despite their advantages, DBMSs are often inefficient for data management in many scientific domains due to the impedance mismatch problem, i.e., the incompatibilities between the representation formats of the source data and the DBMS, as pointed by [9] and [15]. This impedance mismatch yields costly conversions between formats, which adds prohibitive overhead during data analysis. Therefore, scientific file formats such as HDF [39] and NetCDF [31], special libraries, such as Paraview Catalyst [2] and I/O interfaces, such as ADIOS [19] and GLEAN [40] are preferred for maintaining and analyzing scientific datasets. However, although efficient, this low-level approach lacks the powerful analytic capabilities found in a DBMS.

Our previous study on the usage of DBMSs to manage numerical simulation data [22] highlighted many difficulties regarding scientific array data representation. We concluded that the idiosyncrasies of this kind of data are not well represented by current array data models. Also, to perform visualization with data stored in current DBMSs, it is necessary to retrieve the attributes of interest, copying large datasets between memory spaces and carrying out another costly data conversion for the output visualization format. Therefore, as a step towards solving the problem, we proposed an extension of the array data model [23], named TARS (Typed Array Schema), to cope with array data features and to allow a more efficient representation for scientific data.

As the next step in our research agenda to manage scientific array data, we have developed the array DBMS named SAVIME, which implements the TARS data model. SAVIME is an in-memory DBMS designed to analyze complex multidimensional datasets produced by simulations. SAVIME can be coupled to simulations for analysis running in post-processing and in-transit modes [34]. Also, SAVIME incorporates the Paraview Catalyst library, which enables powerful visualization of simulation results, a fundamental tasks performed during scientific simulations.

The multidimensional array data model of SAVIME is an adequate representation for modern machine learning (ML) based applications. Indeed, the use of ML techniques to analyze data both in scientific research and industrial applications is becoming increasingly important, due to their power to capture

patterns represented by complex data correlations that are not easily captured by traditional models, in many different and diverse tasks [7], [8], [10]. Therefore, as pointed out in [29], not only the management but also the understanding of the data becomes a crucial task. The integration of ML-based analytics into the DBMS may lead to powerful performance improvements since different parts of the ML process may be treated as operators of the query plan, providing new opportunities for optimization. Therefore, to cope with the growing need for ML support in scientific research, we have implemented into SAVIME an ML extension through communication with the Tensorflow Extended (TFX) platform [6], integrating the ML process into the DBMS. Such extension enables SAVIME to evaluate any ML algorithm supported by TFX.

To further integrate SAVIME into this process, we have also developed a library that allows the execution of SAVIME queries from the Python environment. These queries results are returned as multidimensional arrays NumPy [27], which are popular structures widely used in scientific data analysis as well as supported by ML libraries and frameworks like PyTorch, scikit-learn, and Tensorflow.

Compared to the current array DBMSs, SAVIME brings the following advantages:

1. Fast data ingestion without impedance mismatch, i.e., the ability to cope with data as it is generated without carrying out costly data conversions during data ingestion.

2. Implementation of a data model that offers an elegant representation for array data and allows SAVIME to take advantage of the preexisting data partitioning and sorting to process queries efficiently

3. Incorporation of the library Paraview Catalyst, which enables the analysis and visualization code to harvest data directly from SAVIME's memory space, avoiding the overhead of creating a separate application to query the DBMS and convert query results to the visualization file format.

4. Integration of the ML process into the DBMS, enabling optimizations that would not be possible when treating ML analytics as independent processes.

In this article, we present SAVIME and the TARS data model, along with a performance evaluation in which we compare SAVIME with SciDB [38], the state-of-the-art array DBMS. We also demonstrate how SAVIME can be easily integrated into the ML process, by presenting a library for communication from a Python environment, and a performance comparison with NumPy arrays.

This paper is an extended version of the work presented in [21], and includes the ML feature not covered in the original. It is organized as follows. In Section 2 we discuss the TARS data model implemented in SAVIME. In Section 3 we present SAVIME, its execution model and its DDL, SDL and DML, along with visualization and ML support. In Section 4 we show the results of our evaluation comparing SAVIME and SciDB and embedding SAVIME with a real application,

and we also present experiments relating to the ML functionality. In Section 5, we discuss the related work and finally in Section 6 we conclude.

2 Typed Array Data Model

Scientific data is usually represented as multidimensional arrays. Multidimensional values are a data pattern that emerges in scientific experiments and measurements and can also be generated by simulations. From a data management and representation perspective, the definition of the array data model is presented in [26]. In short, an array is a regular structure formed by a list of dimensions. A set of indexes for all dimensions identifies a cell or tuple that contains values for a set of array attributes.

If carefully designed, arrays offer many advantages when compared to simple bi-dimensional tables. Cells in an array have an implicit order defined by how the array data is laid out in linear storage. We can have row-major, column-major, or any other arbitrary dimension ordering. Array DBMSs can quickly lookup data and carry out range queries by taking advantage of this implicit ordering. If the data follows a well behaved array-like pattern, using arrays saves a lot of storage space, since dense arrays indexes do not need to be explicitly stored. Furthermore, arrays can be split into subarrays, usually called tiles or chunks. These subarrays are used as processing and storage data units. They help to answer queries rapidly and enforce a coherent multidimensional data representation in linear storage.

However, current array data model implementations, e.g., SciDB [38] and RasDaMan [4], have limitations, preventing an efficient representation of simulation datasets. In SciDB for instance, it might be necessary to preload multidimensional data into a unidimensional array and then rearrange it during data loading. RasDaMan requires either the creation of a script or the generation of compatible file formats for data ingestion. This may also require costly ASCII to binary conversion (since numerical data is likely to be created in binary format) for adjusting the data to the final representation on disk. In both cases, the amount of work for loading the dataset alone is proportional to its size, making it impractical for the multi-terabyte data generated by modern simulation applications.

Furthermore, the array data model does not explicitly incorporate the existence of dimensions whose indexes are non-integer values. In some simulation applications, the data generated follows an array-like pattern, but one of the identifiable dimensions can be a non-integer attribute. For instance, in 3D rectilinear regular meshes, we have points distributed in spatial dimensions whose indexes or coordinate values are usually floating point numbers. To address this issue, we need to map non-integer values into integer indexes that specify positions within the array. Array DBMSs like SciDB or RasDaMan, in their current versions, do not support this kind of functionality.

Arrays can be sparse, meaning that there are no data values for every single array cell. Data may also have some variations in their sparsity from a portion

of the array to another. This is the case for complex unstructured meshes geometry (with an irregular point distribution in space) when directly mapped to arrays. SciDB provides support to sparse arrays, but since it splits an array into chunks (equally sized subarrays), it is very hard to define a balanced partitioning scheme, because data can be distributed very irregularly. RasDaMan is more flexible in this aspect and allows arrays to be split into tiles or chunks with variable sizes.

Another characteristic of complex multidimensional data representation is the existence of partial functional dependencies concerning the set of indexes. Partial dependencies occur in constant or varying mesh geometries and topologies, or any other kind of data that does not necessarily vary along all array dimensions. For instance, when researchers create mathematical and physical models for simulating transient problems, the time is a relevant dimension to all data, i.e., model predictions vary over time. However, the mesh, which is the representation of the spatial domain, may not change in time, meaning that the coordinate values and topology incidence remain the same throughout the entire simulation. Another possibility is the usage of the same mesh for a range of trials, and another mesh for another range. In both cases, there is a mesh for every single time step (an index in the array time dimension) or trial, but actually, only one mesh representation needs to be stored for an entire range of indexes.

Asides from that, simulation and uncertainty quantification applications involve very specific data semantics for various data attributes. These semantical annotations can be incorporated into the data model allowing the definition of special purpose algebraic operators that are useful for creating complex analysis and visualization.

Finally, much effort has been devoted to implementing ML into relational DBMSs, but few array DBMSs implement such support in some way. In RasDaMan, ML models must be implemented as UDFs within the system, being invoked as rasQL queries. In the case of SciDB, the user needs to implement the algorithm using the linear algebra operators provided by the system, but there's no way to easily integrate pre-built ML models as part of the system. This integration could reduce the complexity of scientific analysis and give room to many optimizations, such as query planning, lazy evaluation, materialization, and operator optimization, as pointed out by [11].

2.1 Data Model Overview

In this section, we briefly recall the TARS data model [23] and give a detailed description of how it is implemented in SAVIME.

Given the lack of flexibility in the current array DBMSs in supporting simulation data, as described in Section 2, we propose the TARS data model to cope with the aforementioned issues. A TAR Schema contains a set of Typed ARrays (TARs). A TAR has a set of dimensions and attributes. A TAR cell is a tuple of attributes accessed via a set of indexes. These indexes define the cell location within the TAR. A TAR has a type, formed by a set of roles. A role in

a type defines a special purpose data element with specific semantics.

In TARS (Figure 1), we define mapping functions as a way to provide support for sparse arrays, non-integer dimensions, and heterogeneous memory layouts. With TARS, it is possible to combine array data from different sources, different storage layouts, and even with different degrees of sparsity by associating different mapping functions to different subarrays, or as we call them, subTARs.

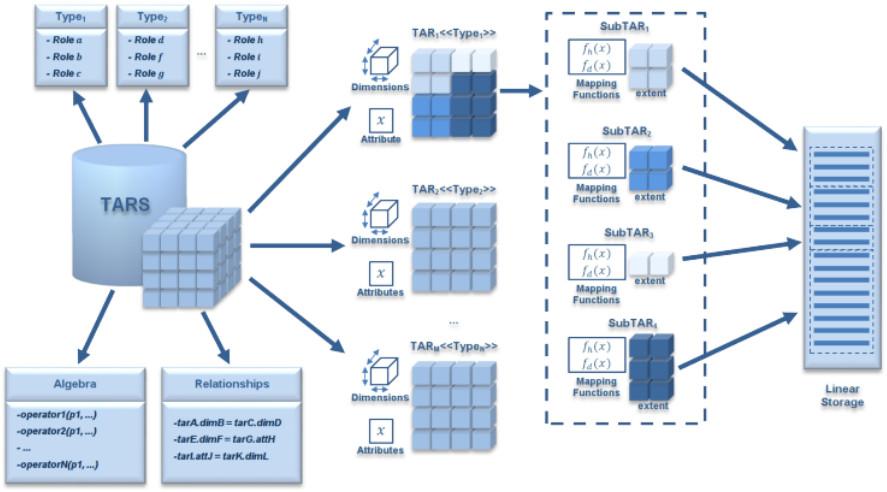

Figure 1: TARS data model

A subTAR covers an n-dimensional slice of a TAR. Every subTAR is defined by the TAR region it represents and two mapping functions: position mapping function and data mapping function. The position mapping function reflects the actual data layout since it defines where every TAR cell within a given subTAR ended in linear storage. Therefore, the position mapping function should implement the multidimensional linearization technique used for the data. The data mapping functions translate a linear address into data values. In a simple scenario, this function does a lookup into a linear array that stores the data. In a more complex scenario, it could compute a derived value from the actual subTAR data.

2.2 Physical Specification

In this section, we describe how the TARS data model is implemented in SAVIME. TARS structures are created in SAVIME with the use of the supported SDL and DDL. Users can define TARs, datasets, and types. Once a TAR is defined and a series of datasets are loaded into the system, it is possible to specify a subTAR by attaching datasets to it. A dataset is a collection of data values of the same type, like a column in a column-store DBMS (SAVIME uses vertical partitioning). A dataset can contain data for a TAR attribute within a

TAR region specified by a subTAR.

TAR dimensions indexes form a domain of values that are represented in SAVIME in two main forms. It can be an implicitly defined range of equally spaced values, in which case, all the user must specify is the lower and upper bounds, and the spacing between two adjacent values. It is called implicit because these indexes do not need to be explicitly stored. For instance, the domain $Di = (0.0, , 2.0, 4.0, 6.0, 8.0, 10.0)$ is defined by the lower bound 0.0, the upper bound 10.0 and all values are equally spaced in 2.0 units.

Dimensions whose indexes do not conform with these constraints have an explicit definition. In this case, the user provides a dataset specifying the dimension indexes values. For instance, consider the following domain $De = (1.2, , 2.3, 4.7, 7.9, 13.2)$. It has a series of values that are not well-behaved and equally spaced, and thus, cannot be represented implicitly.

The data representation within the subTAR requires the combination between the dimension domain and the dimension specifications. All subTARs in a TAR have a list of dimension specifications, one for each dimension in the TAR. These dimension specifications define the TAR region the subTAR encompasses, but they also are a fundamental part in the implementation of the mapping functions. These functions are defined conceptually in the model, but are implemented considering six possible configurations between dimension specifications (ORDERED, PARTIAL and TOTAL) types and dimension types (IMPLICIT and EXPLICIT).

An ORDERED dimension specification indicates that the indexes for the cells in that dimension are dense and sorted in some fashion. A PARTIAL dimension implementation, indicates that there are some holes in the datasets, meaning that some cells at given indexes are not present. Finally the TOTAL representation indicates that data is fully sparse and that all indexes must be given for every cell, in other words, it means that we have a degenerated array that is basically tabular data.

3 The SAVIME system

SAVIME has a component-based architecture common to other DBMSs, containing modules such as an optimizer, a parser, and a query processing engine, along with auxiliary modules to manage connections, metadata and storage. A SAVIME client communicates with the SAVIME server by using a simple protocol that allows both ends to exchange messages, queries, and datasets. All modules are currently implemented as a series of C++ classes, each one of them with an abstract class interface and an underlying concrete implementation.

3.1 Languages DDL, SDL and DML

SAVIME's DDL supports operators to define TARS and Datasets. Listing 1 for instance, presents some of these commands: initially, we issue a `CREATE_TAR` command to create a TAR named `FooTAR`. It has 2 dimensions (I and J) whose

indexes are long integers. These are implicit dimensions, whose domains are integers equally spaced by 1 unit from 1 to 1000. This TAR also has a single attribute named `attrib` whose type is a double precision real number.

```
CREATE_TAR("FooTAR", "*", "Implicit, I, long, 1, 1000, 1 |
                           Implicit, J, long, 1, 1000 , 1",
                           "attrib, double");

CREATE_DATASET("FooBarDS1:double","ds1_data_source");

CREATE_DATASET("FooBarDS2:double","ds2_data_source");

LOAD_SUBTAR("FooTAR", "Ordered, I, 1, 100 |
                       Ordered, J, 1, 100",
                      "attrib, FooBarDS1");

LOAD_SUBTAR("FooTAR", "Ordered, J,101, 200 |
                       Ordered, I, 1, 100",
                      "attrib, FooBarDS2");
```

Listing 1: DDL examples

After that we create 2 datasets named `FooBarDS1` and `FooBarDS2`, they are double typed collections of values in a data source (usually a file or memory-based file). Finally, we issue 2 `LOAD_SUBTAR` commands to create 2 new subtars for the TAR `FooTAR`, the first one encompasses the region that contains the cell s whose indexes are in $[1, 100] \times [1, 100]$ for dimension I and J respectively. In both cases, we have an ordered representation indicating that data is dense and ordered first by the I index and second by J index. The second subtar, however, encompasses the cells whose indexes are in $[1, 100] \times [101, 200]$ but instead ordered first by J index and second by the I index. It is an example of how the SDL works, since users can express and consolidate data sources with different ordering layouts into a single TAR. The final part of the command indicates that datasets `FooBarDS1` and `FooBarDS2` are attached to the `attrib` attribute, meaning that the data for `attrib` in the cells within each subTAR region can be found in these datasets.

SAVIME also supports a functional DML with operators similar to the ones implemented in SciDB, for operations such as filtering data based on predicates, calculating derived values, joins and aggregations. Listing 2 is an example of a query in SAVIME. This DML query consists of three nested operators. Initially, a new attribute called `attrib2` is created by the operator `DERIVE` and its value is defined as the square of the attribute `attrib`. After that, the `WHERE` operator is called, it filters data according to the predicate, in this case, it returns a TAR whose cells have the value for `attrib2` set between 2 and 10. Finally, we use the `AGGREGATE` operator to group data by dimension I indexes and sum the value for `attrib2` creating the sum `attrib2`, the resulting TAR will present only one

```
AGGREGATE(
          WHERE(
                DERIVE(FooTAR, attrib2, attrib*attrib),
                attrib2 >= 2.0 and
                attrib2 <= 10.0
          ),
          sum, attrib2, sum_attrib2, I
);
```

Listing 2: Example of Aggregate Query

dimension (I) and a single attribute sum `attrib2` whose values are the result of the sum of the `attrib2` accross dimension J.

3.2 Catalyst Support and Visualization

Visualization is one of the main activities that scientists execute with data generated by simulations. In this context, viz tools and libraries, such as VTK and Paraview Catalyst [2] are widely used to allow researchers to gain insights about scientific datasets.

SAVIME is able to generate VTK data files to be directed imported by Paraview, or even to execute Catalyst analytical scripts. Visualization in SAVIME is done with the use of a special purpose operator named CATALYZE, shown in Figure 2, which gets the resulting TARs of a query, converts it to the VTK structures and optionally executes a Catalyst script with the query output.

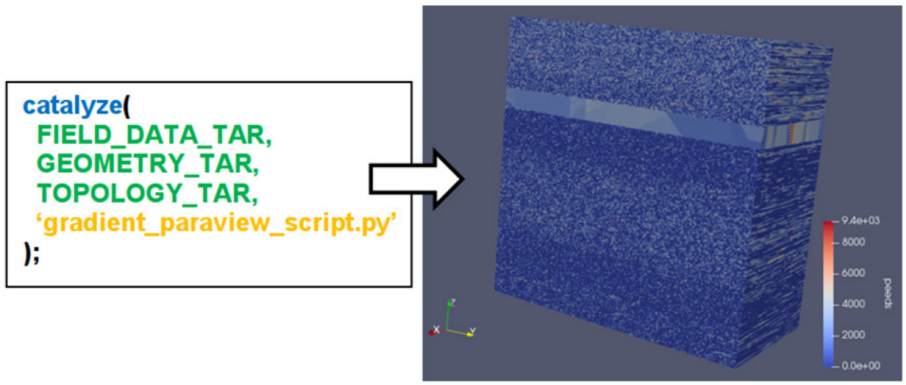

Figure 2: Visualization query using Catalyze operator

In order to implement this operator and visualize data, SAVIME needs to be aware of which parts of the dataset represent the geometry and the topology of the mesh, and how the field data is laid out in this schema and then create

the respective VTK structures. The TARS data model enables this semantic representation by defining special TAR types for topologies, geometries and field data, see Figure 3. For instance, a Cartesian Geometry type has a mandatory role id for identifying points, and other roles for x, y and z coordinates. There is also the same optional roles time step and trial. A mesh geometry can change or evolve during the same trial, in a scenario in which the application domain is deformable, or a different mesh can be used for every simulation run.

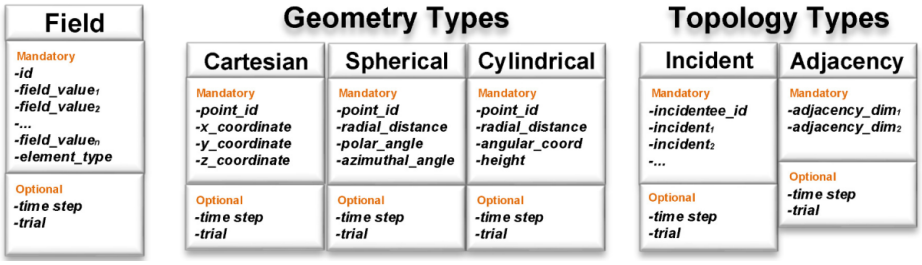

Figure 3: TAR types for data visualization

Data related to the mesh topology requires special types, such as the Incident Topology and Adjacency Topology. The Incident Topology type captures the semantics of topologies specified as a series of incident relationships and the Adjacency topology type captures topologies represented as an adjacency matrix. Since the mesh can be different in every time step for the same run, or for different runs of the same model, the optional roles time step and trial are also present.

Since Catalyst is embedded in SAVIME, it is possible to create visualization files and even run scripts with data harvested directly from SAVIME's memory space. Without such support, users would need to move data out of SAVIME's memory space and convert it before being able to visualize it, which is very inefficient. The SAVIME characteristic presented as an advantage (3) in Section 1 is directly related to this feature implemented in SAVIME.

3.3 Machine Learning Support

Another important aspect of scientific applications is the increasing adoption of ML models for predictions [36][30]. This phenomena points towards extending DBMS systems with the capability of invoking ML models as part of a query expression, so that data preparation and their input to ML models can be jointly optimized. In this context, to provide ML support, SAVIME allows multiple trained models to be registered and used for a given problem. It is also possible to compose the response of different models, evaluated in different regions of the same dataset. To do that, SAVIME counts with a DML operator that receives as input a set of data in an existing TAR and returns as a response, in the form of a new TAR, the result of the execution of a given ML model. This functionality allows the use of models previously defined by the user, simply

by registering them in the system. SAVIME utilizes the Tensorflow Extended platform, a tool for invoking ML models in the form of services. Thus, the extension of SAVIME towards Machine Learning supports any ML algorithm that can be run in the background framework. Future work will extend this feature to other ML frameworks as well. We highlight the fact that because SAVIME only executes trained models, the task of model training must be delegated to other specialized frameworks such as Tensorflow or Pytorch.
In the example depicted in Listing 3, initially, a model is registered in SAVIME

```
REGISTER_MODEL(exampleModel,
                "inputDimA-10|inputDimB-10",
                "outputDimA-5|outputDimB-5",
                "targetAttribute"
);
PREDICT(
        SUBSET(inputTAR,
                inputDimA, 10, 19,
                inputDimB, 10, 19
        ), exampleModel
);
```

Listing 3: Using SAVIME query language to register and invoke a ML model

by using the `REGISTER_MODEL` operator, which receives a model identifier on the system, the input and output dimensions, and a target attribute. Then, by means of the `SUBSET` operator, in a nested query, a slice of the `inputTAR` is selected, according to the specified bounds: 10-19:10-19. The output is finally used as input for the registered model, by using the `PREDICT` operator. We emphasize that the focus of this work is to briefly present SAVIME along with its data model and Machine Learning support. We refer the interested reader to the work of [20] where the details about SAVIME operators and internals are described. In the particular code depicted in Listing 1, the SUBSET is a logical operator, similar to a selection operation in Relational Algebra but enforcing a filtering range condition over dimensions. Its results comprehend a new array with the cells pertaining the dimension range specification.

To provide an easy integration between popular ML analysis tools and SAVIME, we also developed a Python interface named PySAVIME. This interface is a library, written in Python and Cython, that allows the execution of queries in the DBMS from a Python environment. The results of these queries are returned to the environment in the form of multidimensional NumPy arrays. In fact, in order to minimize the overhead caused by the addition of a new layer between the DBMS and the user, the developed library makes extensive use of memory views in Cython. These are envelopes for data buffers that aim to minimize the number of unnecessary copies between data from different programs, in the case of PySAVIME, between the DBMS C++ Client and the library.

Figure 4, to the left, demonstrates a query invoking the `SUBSET` operator into

```
import pysavime.define
import pysavime.operator

def select_tar_subset_query()
  dim_A     = define.implicit_tar_dimension(
                          'dimA', 'int32', 0,
                          ↪  dimA_length)
  dim_B     = define.implicit_tar_dimension(
                          'dimB', 'int32', 0,
                          ↪  dimB_length)

  dimensions = [dim_A, dim_B]
  attribute = define.tar_attribute('attribute',
  ↪  'double')
  attributes_list = [attr_air_temperature]
  tar_definition = define.tar('values_tar',
                           dimensions, attributes)

  query = operator.subset(tar_definition,
                           dim_A.name, 40, 47,
                           dim_B.name, 304, 305)
  return query

with pysavime.Client(host=savime_host,
                     port=savime_port) as client:
  response =
  ↪  client.execute(select_tar_subset_query())
```

DIM_A	DIM_B	Attribute
40	304	298.2
41	304	298.1
42	304	298.1
43	304	298.2
44	304	298.3
45	304	298.0
46	304	298.2
47	304	298.7
40	305	297.6
41	305	296.9
42	305	297.3
43	305	297.6
44	305	297.6
45	305	297.4
46	305	299.0
47	305	298.8

Figure 4: An example of a subset query executed using PySAVIME

SAVIME, by using the PySAVIME library. The `tar_definition`, contains information about the name of the target-tar, its dimensions and attribute values, and is is created using the PySAVIME command `define`. The operator subset command is defined using the PySAVIME class `Operator`. Finally, the query can be run using the command `client.execute`. Once executed, its result is returned as a NumPy array, as illustrated in Figure 4, to the right.

3.4 Query Processing

As presented in the previous section, a SAVIME query contains a series of nested operators. Most of them expect one or more input TARs and originate a newly created output TAR. Unless a special operator is called to materialize the query resulting TAR, it is generated as a stream of subTARs, sent to the client, and then discarded. SAVIME operates TARs as a subTARs stream pipelined across operators. SubTARs are processed serially or in parallel with OpenMP constructs.

During query processing, when a subTAR for a TAR holding intermediated results is generated and passed on to the next operator, it is maintained in a temporary subTARs cache. These subTARs contain their own group of datasets that could require a lot of storage space or memory. Therefore, once a subTAR is no longer required by any operator, it must be removed from memory. An operator implementation is agnostic regarding its previous and posterior operations in the pipeline and does not know when to free or not a subTAR. All the operators' implementation needs to establish when it will no longer require a given subTAR. When this happens, the operator notifies the execution engine that it is done with a given subTAR and it is then discarded

However, since the same subTAR can potentially be input into more than one operator during a query, freeing it upfront is not a good idea, because it might be required again. In this case, SAVIME would have to recreate it. To solve this problem, every subTAR has an associated counter initially set to the number of operators that have its TAR as their input. When an operator notifies the engine that it no longer needs that specific subTAR, the respective counter is decreased. Once the counter reaches zero, all operators possibly interested in the subTAR are done, and now it is safe to free the subTAR. This approach always frees the used memory as soon as possible and never requires a subTAR to be created twice. However, some operators might require many subTARs to be kept in memory before freeing them. In an environment with limited memory, it would not be feasible to cope with very large TARs in this case. A solution then would be the adoption of a more economical approach, trading off space with time by freeing and regenerating the subTARs whenever memory is running low.

4 Experimental Evaluation

We ran a series of experiments in order to validate SAVIME as an efficient system for simulation data management. First, we compare SAVIME with SciDB and evaluate how SAVIME affects the performance of the actual simulation code. Then we demonstrate how SAVIME can be a feasible tool integrated into the ML process by comparing its execution time with the equivalent procedure using NumPy arrays.

4.1 SAVIME vs SciDB

In this section, we compare SAVIME, SciDB (version 16.9), and a third approach based on the usage of NetCDF files (version 4.0). We provide a NetCDF comparison only as a baseline, since it does not offer all benefits that a DBMS does (see Section 5). All scripts, applications, and queries used in our evaluation are available at `github.com/hllustosa/savime-testing`.

We employ two datasets, a dense and a sparse one, based on data from the HPC4e BSC seismic benchmark [12] in our experiments. The dense dataset contains 500 trials (one dataset for each) for a 3D regular mesh with dimensions 201x501x501 containing a velocity field. In total, we have over 30 billion array cells and more than 120 GB of data. All data is held in a single 4D structure (TAR in SAVIME, array in SciDB and in a single NetCDF file) containing the X, Y, and Z dimensions, and an extra trial dimension to represent all 500 trials. Both structures have the same tiling configuration, i.e., the same number of chunks/subTARs (500 of them, one for each trial) with the same extents.

The sparse dataset is also a 4D structure with 500 simulation trials, but only a subset of the cells are present (around 24% of the dense dataset). It comprises almost 8 billion array cells and over 30 GB of data. We used a sparse 4D array/TAR in SciDB and SAVIME, and a 2D dense array in NetCDF. NetCDF lacks the ability to natively represent sparse data, thus we indexed the x, y, and z values and represented them as a single dimension and stored coordinate values as variables.

The computational resource used is a fatnode from the cluster Petrus at DEXLab. This fatnode has 6 Intel(R) Xeon(R) CPU E5-2690 processors amounting to 48 cores and over 700 GB of RAM. Data is kept in a shared-memory file system to simulate an in-transit data analysis, in which data is not kept on disk (for both SAVIME and SciDB). Initially, we evaluate the loading time of 500 tiles/chunks in all three approaches, considering that data is being transferred and continually appended to a single array/TAR/file as it is being generated by a solver.

As we can see in Figure 5 on the left graph, the ingestion time taken by SciDB is almost 20 times longer than the time taken by SAVIME, due to costly rearrangements needed on data to make it conform with the underlying storage configuration. Besides, there is an extra overhead during the lightweight data compression done by SciDB, which makes the dataset roughly 50% smaller when stored but increased loading time prohibitively. In contrast, SAVIME does not alter or index the data during the process of data ingestion, therefore the loading process is computationally much cheaper.

We evaluate the performance considering ordinary and exact window queries. Ordinary Window queries consist of retrieving a subset of the array defined by a range in all its dimensions. The performance for this type of query depends on how data is chunked and laid out. High selectivity queries, which need to retrieve only a very small portion of the data tends to be faster than full scans. Therefore, we compared low and high selectivity queries, filtering from a single to all 500 tiles. We also considered the performance of exact window queries,

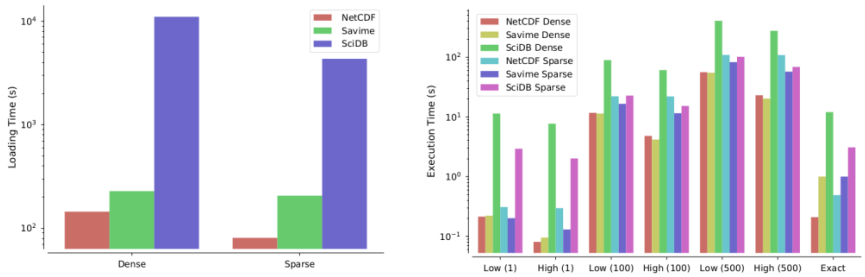

Figure 5: Ingestion and query execution time

which is the easiest type of Window Query. They consist of retrieving data for a single tile or chunk, meaning the system has close to zero work filtering out the result.

We implement these queries as specific operators in SAVIME and SciDB and with the help of a custom OpenMP application using the NetCDF library. The experimental results are shown in Figure 5 on the right graph. The average time of 30 runs is presented. We considered window queries with low selectivity (over 70% of all cells in a tile) and high selectivity (around 20% of all cells in a tile), and intersecting with only 1, 100, or even the total 500 tiles.

It is noticeable that SAVIME either outperforms SciDB or is as efficient as it in all scenarios. The most important observation is that, even without any previous data preprocessing, SAVIME is able to simply take advantage of the existing data structure to answer the queries efficiently, which validates the model as a feasible alternative to existing implementations. The results show that, for any storage alternative in both dense and sparse formats, the subsetting of a single tile is very efficient. The differences shown for the exact window query and for low and high selectivity window queries that touch a single tile are very small. SciDB takes a few seconds in most cases, while SAVIME takes on average 1 second. NetCDF is the most efficient in this scenario, retrieving desired data in less than a second.

However, for queries touching 100 or 500 chunks, we can see the differences between querying dense and sparse arrays. The dense dataset is queried more efficiently since it is possible to determine the exact position of every cell and read only the data of interest. It is not possible for sparse data since one is not able to infer cell positions within the tiles. In this case, every single cell within all tiles that intersect with the range query must be checked

In dense arrays, we can observe a reduced time for retrieving data in high selectivity queries in comparison with low selectivity queries. The execution time of window queries should depend only on the amount of data of interest since cells can be accessed directly and thus, no extra cells need to be checked. The execution times considering 100 or 500 tiles in SAVIME and NetCDF are in accordance with this premise. However, SciDB shows poorer performance, being up to 8 times slower. It is very likely that SciDB needs to process cells outside of

the window of interest depending on the compression technique and the storage layout adopted. SciDB seems to be more sensible to tiling granularity, requiring fine-grained tiles that match the window query to have a performance similar to the NetCDF approach.

There is not much to be done for querying sparse arrays except for going through every cell in the tiles intersecting the window specified by the query. The query time for sparse data in all alternatives shows very similar performance. The main difference is that for achieving this result with NetCDF, an OpenMP application needed to be written, while the same result could be obtained with a one-line query in SAVIME and SciDB.

Our conclusion is that the regular chunking scheme imposed by SciDB not only slows down the ingestion process significantly as it has no real impact in improving performance for simple operations as SAVIME using a more flexible data model can solve similar queries presenting a compatible performance.

4.2 Integration with a numerical solver

In this section, we evaluate the amount of overhead imposed on the simulation code when integrating with SAVIME. We use the simulation tools based on the MHM numerical method [14] as a representative numerical simulation application. We compare three approaches. In the first approach, SAVIME is used IN-TRANSIT, in a single node (fatnode) while the simulation code runs in a different set of nodes, and thus data needs to be transferred. In the second approach, SAVIME is used IN-SITU, with individual SAVIME instances running on each node, the same used by the simulation code. In this scenario, the data does not need to be transferred, since it is maintained in a local SAVIME instance that shares the same computational resources used by the simulation code. In the third approach, SAVIME is not used, but instead, the data is stored in ENSIGHT files (the standard file format used by MHM), and analyzes are performed by an ad-hoc Message Passing Interface application in Python. This last scenario serves as a baseline implementation, thus we call it the baseline approach. The computational resource used is the Petrus cluster at DEXLab, with 8 nodes, each with 96 GB of RAM and 2 Intel(R) Xeon(R) CPU E5-2690 processors.

Preliminarily, we start the evaluation by measuring the overhead of loading data in a remote node running SAVIME. In Figure 6 we can see the time of running the simulation and discarding the data, which is the time to solely carry out the computations without any I/O whatsoever and the time to transfer and load data into SAVIME. We vary the size of the MHM meshes, which impact on the level of detail of the simulation. The larger the mesh size is, the more realistic and complex the simulation is, and also, more resources (time and memory) are consumed.

Results show that for very small meshes, which are computationally cheap, the time to load data is more significant, and thus there is some overhead (around 40%) in storing data in SAVIME. However, as meshes get larger, the time taken to compute them increases in a manner that the transfer and load-

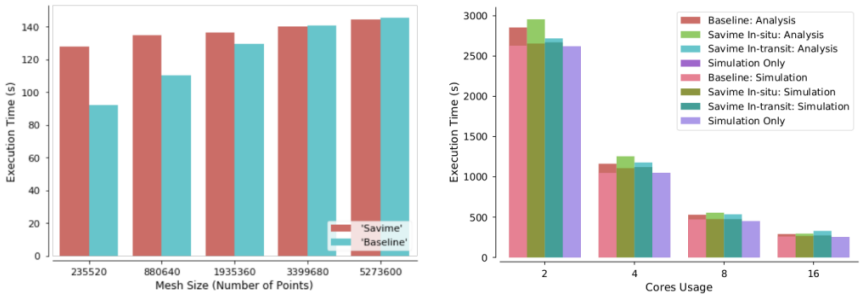

Figure 6: Mesh size x Simulation Execution Time (left) and Simulation and Analysis Scalability (right)

ing time is negligible in comparison to the time taken to compute the solution. The transfer and loading time is masked by the dominant process of solving the systems of equation. Therefore, this shows that, for large enough problems, it is possible to load data into SAVIME without compromising the simulation code performance. However, under an impedance mismatch scenario, as observed with SciDB, the loading time would be significant, impacting on the simulation cost.

To evaluate the integration between SAVIME and a simulation tool based on the MHM solver, we use a 2D transport problem over a mesh with 1.9 million points. We run the simulation up to the 100th time step, and store either in SAVIME (approaches 1 and 2) or in an ENSIGHT file (approach 3), data from 50% of all the computed time steps. In both cases, data is always kept in a memory based file system, and never stored on disk. Once data has been generated, it is analyzed with a PARAVIEW pipeline that carries out the computation of the gradient of the displacement field of the solution. This part is either done by a special operator in SAVIME, or by an AD-HOC MPI Python application using the Catalyst library (baseline), depending on the approach being run. Additionally, we measure the cost of running the simulation without any further analysis, to highlight the simulation only cost

Figure 5 shows the results when running the three approaches, varying the amount of MPI processes spawned or the number of cores used by the MHM simulation code. In this experiment, the simulation code runs and produces its results, and then, the simulation output data is read and processed in the analysis step. The plot shows, for each evaluated number of MPI processes, the simulation time and the analysis time as stacked bars. The graph shows that the cost of the analysis process is significantly smaller than the cost of computing the simulation. Moreover, as the Simulation Only run shows, the overhead introduced by storing the data in SAVIME or as an ENSIGHT file is negligible, which confirms the claim that SAVIME can be introduced into the simulation process without incurring in extra overhead. From the point of view of the effect of SAVIME on simulation scalability, the storage of data in

SAVIME does not impair the capability of the simulation code to scale up to 16 cores.

The IN-TRANSIT approach differs from the other two approaches since it uses a separate computational resource to execute the analysis step. As we see in Figure 6, even when we increase the number of cores the simulation code uses, the analysis time does not change, because the analysis step is done in the fatnode, and always uses the same number of cores (16) independently from the actual number of cores used by the simulation code. The IN-TRANSIT approach illustrates a scenario in which all data is sent to a single computational node and kept in a single SAVIME instance. This approach offers some extra overhead and contention since all data is sent to a single SAVIME instance, but this enables posterior analysis that transverse the entire dataset without requiring further data transfers.

The SAVIME IN-SITU approach uses the same computational resources used by the simulation code. When we increase the number of cores used by the simulation code, we also increase the numbers of cores used by SAVIME for analysis. The same is true for the baseline approach, meaning that the AD-HOC application also uses the same number of cores the simulation code uses. Even though the SAVIME IN-SITU approach is slightly slower than the baseline approach, we see that both are able to scale similarly. The difference observed in performance between using SAVIME and coding a specialized application becomes less significant as we increase the number of cores being used during the analysis phase. Nevertheless, the small performance loss in this case might be justified by the convenience of using a query language to express analysis instead of the extensive and error prone process of coding other applications to execute the analysis.

4.3 Integration with Tensorflow Extended

To demonstrate how SAVIME is a viable alternative in terms of performance when applied to the ML process, we compared the execution time of a prediction query run in the system with the equivalent procedure invoking Tensorflow Extended from a Python environment using only NumPy arrays. In our experiment, we used NumPy version 1.18.5, and Python version 3.6.9. To perform these experiments, we used a subset of the Climate Forecast System Reanalysis (CFSR) dataset [33] containing air temperature observations registered every 6 hours from January 2010 to December 2012, with a spatial resolution of 0.5 degrees.

To perform the predictions, we used three spatio-temporal deep learning models whose inputs are a list of fixed size frames. Given an input, the model's output is a list of frames of the same size as the input. The models have an input frame spatial size of 10x10 for model A, 10x20 for model B, and 10x30 for model C, and were trained in different regions of the dataset. The architecture we adopted in the experiments is the ConvLSTM.

As the input size of the models differs, we performed three different input queries, selecting a range of the dataset corresponding to the coordinates of

5S:14S, 62W:53W for the first model A, 5S:14S, 62W:43W for the second one B and 5S:14S, 62W:33W to the last model C.

To invoke TFX, the models' input must first be converted to a JSON format and sent to the server. After the model is executed, the response must also be converted back to the original format. We performed the experiments 10 times and evaluated the average runtime of each of these steps, both in SAVIME and by using NumPy arrays.

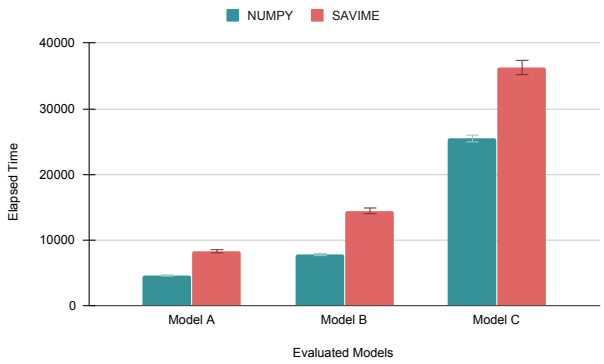

Figure 7: Execution time of the prediction query using SAVIME and using NumPy (in milliseconds)

The summary of our experimental results can be seen in Figure 7, and in Figure 8. In Figure 7, we can see that the total execution time of the prediction query when using SAVIME is 1.8, 1.85, and 1.4 times the total execution time of the equivalent operation when using NumPy, for models A, B, and C respectively. In Figure 8 we can see the time taken to execute each step.

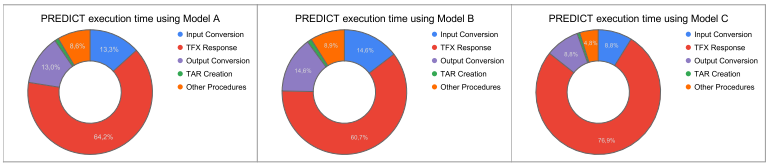

Figure 8: Detailed execution time of the prediction query in SAVIME for three different models

While the process using SAVIME still incurs some overhead when compared to the same process when using NumPy arrays, this overhead can be compensated by different optimizations that become possible by integrating the ML model as part of the DBMS. Furthermore, as already demonstrated in previous sections, the ingestion time when using SAVIME is very much faster than using other array DBMSs, being only a subset of the dataset necessary to run the query.

5 Related Work

The definition of the first array data models and query languages dates back to the works of Baumann [3], and Marathe [24] [25]. The oldest known array DBMS is RasDaMan [5], which adds complete array queries capabilities on top of a relational DBMS. Nowadays, the most popular array DBMS implementation is SciDB [13]. However, both array models implemented of RasDaMan and SciDB have limitations that hamper the representation of scientific datasets, like those originated by simulations.

Due to the fact that ingesting data into these systems is not an easy task, other arrays systems, such as ArrayBridge [41] and ChronosDB [42] have been developed. ArrayBridge works over SciDB, and gives it the ability to work directly with HDF5 files. ChronosDB works over many file formats in the context of raster geospatial datasets. SAVIME has a similar goal to ease data ingestion, however, it does so by enabling seamless data ingestion considering many different array data source by supporting a SDL and a flexible data model, which makes it different from ArrayBridge. SAVIME also offers its own DML, while ChronosDB makes use of existing applications to process data. SAVIME is also different from TileDB [28], which is not exactly a DBMS with a declarative query language, but a library that deals with array data. In addition, SAVIME is a specialized in-memory solution, while the rest of these systems are usually more disk oriented solutions

Another issue is related to the impedance mismatch problem. The underlying data models and storage schemes of a DBMS might differ greatly from the source format the data is initially generated, which forces users to convert their datasets to a compatible representation in order to load them into the system. This problem is common in scientific application domains. For instance, loading and indexing simulation datasets into a DBMS can incur high overhead, thus preventing scientists from adopting any DBMS at all. Instead, scientists typically rely on I/O libraries that provide support for multidimensional arrays, e.g., HDF [39]. and NetCDF [31]. These libraries give users more control over their data without incurring the performance penalties for data loading in a DBMS [9] [15]. They are also flexible, allowing users to specify how their data (produced by simulations) is laid out and avoid the expensive conversions performed by the DBMS during data ingestion.

However, I/O libraries do not offer all benefits that a full-fledged DBMS does. These benefits include, but are not limited to features, such as query languages, data views, and the isolation between data and applications that consume data. NoDB [1] is a first attempt to bridge the gap between DBMS high ingestion costs and I/O libraries access efficiency for relational DBMSs. The approach advocates that the DBMS should be able to work with data as laid out by the data producer, with no overhead for data ingestion and indexing. Any subsequent data transformation or indexing performed by the DBMS in order to improve the performance of data analyses should be done adaptively as queries are submitted. We believe that even though NoDB is currently implemented on top of a RDBMS and lacks support for multidimensional data, its philosophy

can be successfully applied for array databases as well. In this context, SAVIME provides means for fast data ingestion free from costly data conversions common to other array DBMSs. The TARS data model allows huge memory chunks that contain arrays of numerical values output from solvers of linear systems to be efficiently ingested into the system without any type of rearrangements.

From a data model perspective, the challenges of representing simulation data efficiently have been tackled in many works [32] [16] [17]. Some particular data models have been proposed, but none of them was established as a standard for simulation data. In general, arrays are considered the most common format for scientific data. Scientific databases and the aforementioned I/O libraries support arrays. Even scientific visualization libraries, such as VTK [35], which allows the most varied grids to be represented and visualized, has arrays (called VTKArrays) as its underlying data structure.

In SAVIME, the idiosyncrasies of simulation data are captured through the usage of specially defined arrays with associated semantical annotations. These or typed arrays conform with a defined type that represents some aspect of grid data, such as geometries topologies or field data.

6 Conclusion

The adoption of scientific file formats and I/O libraries rather than DBMSs for scientific data analysis is due to a series of problems concerning data representation and data ingestion in current solutions. To mitigate these problems, and to also offer the benefits of declarative array processing in memory, we propose a system called SAVIME. We showed how SAVIME, by implementing the TARS data model, does not impose the huge overhead present in current database solutions for data ingestion, while also being able to take advantage of preexisting data layouts to answer queries efficiently. We also showed SAVIME's ML support, demonstrating how it is a viable tool on the ML process.

We compared SAVIME with SciDB and a baseline approach using the NetCDF platform. The experimental results show that SciDB suffers from the aforementioned problems, not being an ideal alternative and that SAVIME enables faster data ingestion while maintaining similar performance during window queries execution. We showed that SAVIME can also match the performance of NetCDF for loading and querying dense arrays while providing the benefits of a query language processing layer.

We also assess SAVIME's performance when integrating with simulation code. In this evaluation, we showed that storing data in SAVIME does not impair the scalability of the solver. In addition, results also show that it is possible to retrieve SAVIME data and generate viz files efficiently by using the special purpose visualization operator.

SAVIME is available at `github.com/dexllab/Savime`. Future work might focus on the improvement and optimization of current operators and on the development of new special purpose TAR operators. It might also focus on the ML feature. Since SAVIME treats the ML prediction as an operator of the query

plan, there is a vast opportunity for optimizations that can be implemented in the system, e.g. optimizing the query plan, performing lazy evaluation.

Acknowledgments

The authors would like to thank Yania Souto, for the ML models used on the experiments, as well as Brian Tsan and Florin Rusu from the University of California, Merced, for contributions to the ML feature implementation in SAVIME.

References

[1] Ioannis Alagiannis et al. "NoDB: efficient query execution on raw data files". In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. Arizona, USA: Association for Computing Machinery, 2012, pp. 241–252.

[2] Utkarsh Ayachit et al. "Paraview catalyst: Enabling in situ data analysis and visualization". In: *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*. Texas, USA: Association for Computing Machinery, 2015, pp. 25–29.

[3] Peter Baumann. "Management of multidimensional discrete data". In: *The VLDB Journal* 3.4 (1994), pp. 401–444.

[4] Peter Baumann et al. "The multidimensional database system RasDaMan". In: *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. Washington, USA: Association for Computing Machinery, 1998, pp. 575–577.

[5] Peter Baumann et al. "The RasDaMan approach to multidimensional database management". In: *Proceedings of the 1997 ACM symposium on Applied computing*. California, USA: Association for Computing Machinery, 1997, pp. 166–173.

[6] Denis Baylor et al. "Tfx: A tensorflow-based production-scale machine learning platform". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Nova Scotia, Canada: Association for Computing Machinery, 2017, pp. 1387–1395.

[7] Lucas O Bayma and Marconi Arruda Pereira. "Identifying finest machine learning algorithm for climate data imputation in the state of Minas Gerais, Brazil". In: *Journal of Information and Data Management* 9.3 (2018), pp. 259–259.

[8] Karin Becker, Jonathas Gabriel Harb, and Regis Ebeling. "Exploring deep learning for the analysis of emotional reactions to terrorist events on Twitter". In: *Journal of Information and Data Management* 10.2 (2019), pp. 97–115.

[9] Spyros Blanas et al. “Parallel data analysis directly on scientific file formats”. In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. Utah, USA: Association for Computing Machinery, 2014, pp. 385–396.

[10] Dieinison Jack Freire Braga et al. “Time Series Forecasting to Support Irrigation Management”. In: *Journal of Information and Data Management* 10.2 (2019), pp. 66–80.

[11] Shaofeng Cai et al. “Model slicing for supporting complex analytics with elastic inference cost and resource constraints”. In: *Proceedings of the VLDB Endowment* 13.2 (2019), pp. 86–99.

[12] Barcelona Supercomputing Center. *New hpc4e seismic test suite to increase the pace of development of new modelling and imaging technologies*. online. 2016.

[13] Philippe Cudré-Mauroux et al. “A demonstration of SciDB: a science-oriented DBMS”. In: *Proceedings of the VLDB Endowment* 2.2 (2009), pp. 1534–1537.

[14] Antônio Tadeu A. Gomes et al. “On the Implementation of a Scalable Simulator for Multiscale Hybrid-Mixed Methods”. In: *CoRR* abs/1703.10435 (2017). arXiv: `1703.10435`. URL: `http://arxiv.org/abs/1703.10435`.

[15] Luke Gosink et al. “HDF5-FastQuery: Accelerating complex queries on HDF datasets using fast bitmap indices”. In: *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*. IEEE. Vienna, Austria: IEEE Computer Society, 2006, pp. 149–158.

[16] Bill Howe. “Gridfields: Model-driven Data Transformation in the Physical Sciences”. PhD thesis. USA: Portland State University, 2006.

[17] Byung S Lee et al. *Modeling and querying scientific simulation mesh data*. 2002.

[18] Alexandre AB Lima, Marta Mattoso, and Patrick Valduriez. “Adaptive virtual partitioning for OLAP query processing in a database cluster”. In: *Journal of Information and Data Management* 1.1 (2010), pp. 75–75.

[19] Jay Lofstead et al. “Adaptable, metadata rich IO methods for portable high performance IO”. In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE. Los Alamitos, California, USA: IEEE Computer Society, 2009, pp. 1–10.

[20] Hermano Lustosa. “SAVIME: Enabling Declarative Array Processing In Memory”. PhD thesis. Petrópolis, Brazil: Laboratório Nacional de Computação Científica, 2020.

[21] Hermano Lustosa, Fabio Porto, and Patrick Valduriez. “SAVIME: A Database Management System for Simulation Data Analysis and Visualization”. In: *Proceedings of the Brazilian Symposium on Databases*. Vol. 34. Ceará, Brazil, 2019, pp. 85–96.

[22] Hermano Lustosa et al. “Database system support of simulation data”. In: *Proceedings of the VLDB Endowment (PVLDB)* 9.13 (2016), pp. 1329–1340.

[23] Hermano Lustosa et al. “Tars: An array model with rich semantics for multidimensional data”. In: *Proceedings of the ER Forum 2017 and the ER 2017 Demo Track co-located with the 36th International Conference on Conceptual Modelling (ER 2017), Valencia, Spain, - November 6-9, 2017*. Vol. 1979. CEUR Workshop Proceedings. Valencia, Spain: CEUR-WS.org, 2017, pp. 114–127.

[24] Arunprasad P Marathe and Kenneth Salem. “A language for manipulating arrays”. In: *VLDB*. Vol. 97. Tucson, Arizona, USA: Association for Computing Machinery, 1997, pp. 46–55.

[25] Arunprasad P Marathe and Kenneth Salem. “Query processing techniques for arrays”. In: *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1999, pp. 323–334.

[26] Arunprasad P Marathe and Kenneth Salem. “Query processing techniques for arrays”. In: *The VLDB Journal* 11.1 (2002), pp. 68–91.

[27] Travis E Oliphant. *A guide to NumPy*. Vol. 1. USA: Trelgol Publishing, 2006.

[28] Stavros Papadopoulos et al. “The TileDB array data storage manager”. In: *Proceedings of the VLDB Endowment* 10.4 (2016), pp. 349–360.

[29] Eduardo Pena et al. “Mind Your Dependencies for Semantic Query Optimization”. In: *Journal of Information and Data Management* 9.1 (2018), pp. 3–3.

[30] Petrillo. “Finding strong gravitational lenses in the Kilo Degree Survey with convolutional neural networks”. In: *Monthly Notices of the Royal Astronomical Society* 472.1 (2017), pp. 1129–1150.

[31] Russ Rew and Glenn Davis. “NetCDF: an interface for scientific data access”. In: *IEEE computer graphics and applications* 10.4 (1990), pp. 76–82.

[32] Alireza Rezaei Mahdiraji, Peter Baumann, and Guntram Berti. “Img-complex: graph data model for topology of unstructured meshes”. In: *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. San Francisco, California, USA: Association for Computing Machinery, 2013, pp. 1619–1624.

[33] Suranjana Saha et al. “The NCEP climate forecast system reanalysis”. In: *Bulletin of the American Meteorological Society* 91.8 (2010), pp. 1015–1058.

[34] Allan Santos et al. “Towards In-transit Analysis on Supercomputing Environments”. In: *CoRR* abs/1805.06425 (2018). arXiv: 1805.06425. URL: http://arxiv.org/abs/1805.06425.

[35] W Schroeder, K Martin, and B Lorensen. "The visualization toolkit, 4th edn. Kitware". In: *New York* (2006).

[36] Xingjian Shi et al. "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting". In: *Proceedings of 2015 NIPS International Conference on Neural Information Processing Systems*. 2015, pp. 802–810.

[37] Michael Stonebraker et al. "One size fits all? Part 2: Benchmarking results". In: *Proc. CIDR*. California, USA: CIDR Conference, 2007.

[38] Michael Stonebraker et al. "SciDB: A database management system for applications with complex analytics". In: *Computing in Science & Engineering* 15.3 (2013), pp. 54–62.

[39] The HDF Group. *Hierarchical Data Format, version 5*. http://www.hdfgroup.org/HDF5/. 1997-2020.

[40] Venkatram Vishwanath et al. "Topology-aware data movement and staging for I/O acceleration on Blue Gene/P supercomputing systems". In: *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. Washington, USA: Association for Computing Machinery, 2011, pp. 1–11.

[41] Haoyuan Xing et al. "ArrayBridge: Interweaving declarative array processing in SciDB with imperative HDF5-based programs". In: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE. 2018, pp. 977–988.

[42] Ramon Antonio Rodriges Zalipynis. "Chronosdb: distributed, file based, geospatial array dbms". In: *Proceedings of the VLDB Endowment* 11.10 (2018), pp. 1247–1261.