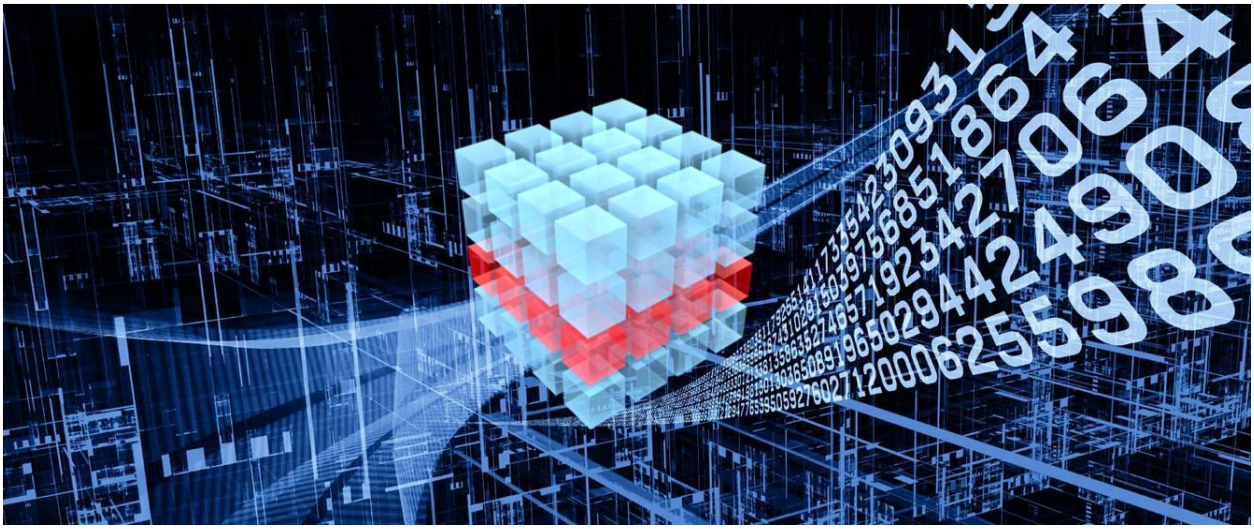

Array Databases: Concepts, Standards, Implementations

Peter Baumann[1], Dimitar Misev[1], Vlad Merticariu[1], Bang Pham Huu[1], Brennan Bell[1], Kwo-Sen Kuo[2]

[1] Jacobs University
Large-Scale Scientific Information Systems Research Group
Bremen, Germany
{p.baumann,d.misev,v.merticariu,b.phamhuu,b.bell}@jacobs-university.de

[2] Bayesics, LLC / NASA
USA
kwo-sen.kuo@nasa.gov

Corresponding author: Peter Baumann, p.baumann@jacobs-university.de

This report is proposed for an official RDA Recommendation
produced by the RDA Array Database Assessment Working Group.

Executive Summary

Multi-dimensional arrays (also known as raster data or gridded data) play a core role in many, if not all science and engineering domains where they typically represent spatio-temporal sensor, image, simulation output, or statistics "datacubes". However, as classic database technology does not support arrays adequately, such data today are maintained mostly in silo solutions, with architectures that tend to erode and have difficulties keeping up with the increasing requirements on service quality.

Array Database systems attempt to close this gap by providing declarative query support for flexible ad-hoc analytics on large n-D arrays, similar to what SQL offers on set-oriented data, XQuery on hierarchical data, and SPARQL or CIPHER on graph data. Today, Petascale Array Database installations exist, employing massive parallelism and distributed processing. Hence, questions arise about technology and standards available, usability, and overall maturity.

To elicit the state of the art in Array Databases, Research Data Alliance (RDA) has established the Array Database Assessment Working Group (ADA:WG) as a spin-off from the Big Data Interest Group, technically supported by IEEE GRSS. Between September 2016 and March 2018, the ADA:WG has established an introduction to Array Database technology, a comparison of Array Database systems and related technology, a list of pertinent standards with tutorials, and comparative benchmarks to essentially answer the question: *how can data scientists and engineers benefit from Array Database technology?*

Investigation shows that there is a lively ecosystem of technology with increasing uptake, and proven array analytics standards are in place. Tools, though, vary greatly in functionality and performance as investigation shows. While systems like rasdaman are Petascale proven and parallelize across 1,000+ cloud nodes, others (like EXTASCID) still have to find their way into large-scale practice. In comparison to other array services (MapReduce type systems, command line tools, libraries, etc.) Array Databases can excel in aspects like service friendliness to both users and administrators, standards adherence, and often performance. As it turns out, Array Databases can offer significant advantages in terms of flexibility, functionality, extensibility, as well as performance and scalability – in total, their approach of offering "datacubes" analysis-ready heralds a new level of service quality. Consequently, they have to be considered as a serious option for "Big DataCube" servicees in science, engineering and beyond.

The outcome of this investigation, a unique compilation and in-depth analysis of the state of the art in Array Databases, is supposed to provide beneficial insight for both technologists and decision makers considering "Big Array Data" services in both academic and industrial environments.

Table of Contents

1 Introduction

"The speed at which any given scientific discipline advances will depend on how researchers collaborate with one another, and with technologists, in areas of eScience such as databases[...]" -- The Fourth Paradigm

1.1 Why Do We Need "Arrays"?

The significant increase in scientific data that occurred in the past decade – such as NASA's archive growth from some hundred Terabytes in 2000 [24] to 32 Petabytes of climate observation data [45], as well as ECMWF's climate archive of 220 Petabytes [22] – marked a change in the workflow of researchers and programmers. Early approaches consisted mainly of retrieving a number of files from an FTP server, followed by manual filtering and extracting, and then either running a batch of computation processes on the user's local workstation, or tediously writing and optimizing sophisticated single-use-case software designed to run on expensive supercomputing infrastructures. This is not feasible any more when dealing with Petabytes of data which need to be stored, filtered and processed beforehand. When data providers discovered this they started providing custom tools themselves, often leading to silo solutions which turn out to erode over time and make maintenance and evolution hard if not impossible. An alternative finding attention only recently are database-centric approaches, as these have shown significant potential; meantime, we find both small institutions [35] and large data-centers [22] using modern database architectures for massive spatio-temporal data sets.

Arrays - also called "raster data" or "gridded data" or, more recently, "datacubes" [8] - constitute an abstraction that appears in virtually all areas of science and engineering, and even beyond:

- **Earth sciences:** 1-D sensor data, 2-D satellite imagery, 3-D x/y/t image timeseries and x/y/z subsurface voxel data, 4-D x/y/z/t climate and weather data; etc.
- **Life sciences:** microarray data, image modalities like X-ray, sonography, PET, and fMRI delivering 2-D and increasingly 3-D data about human and non-human brains and further organs; 2-D through 4-D gene expression data; etc.
- **Space sciences:** optical and radio telescope data; 4-D x/y/z/t cosmological simulation output; planetary surface and subsurface data; etc.
- **Statistics:** "Datacubes" are known since long in the context of Data Warehousing and OLAP [15] where, instead of spatial, abstract axes are defined, usually together with a time axis. A main difference to the above data is that statistical datacubes are rather sparse (say, 3% - 5% of the data space is occupied by values) whereas Earth, Space, and Life science and engineering data tend to be rather dense, often completely dense (i.e., most or all cell positions in the grid hold some non-null value).

Fig. 1 gives an impression of the variety of different observed data specifically in Ocean science. Generally, arrays typically represent sensor, image, simulation, and statistics data of spatio-temporal or "abstract" dimensions.

Fig. 1. Integrating a variety of data sources and types in an Ocean Science Interoperability Experiment (source: OGC membership)

1.2 Why Array Databases?

For decades now, SQL has proven its value in any-size data services in companies as well as public administration. Part of this success is the versatility of the query lanuage approach, as well as the degree of freedom for vendors to enhance performance through server-side scalability methods. Unfortunately, scientific and engineering environments could benefit only to a limited extent. The main reason is a fundamental lack in data structure support: While flat tables are suitable for accounting and product catalogues, science needs additional information categories, such as hierarchies, graphs, and arrays. The consequence of this missing support has been a historical divide been ”data” (large, constrained to download, no search) and ”metadata” (small, agile, searchable).

Still, databases have worked out some key components of a powerful, flexible, scalable data management; these principles have proven successful over decades[1] on sets (relational DBMSs), hierarchical data (e.g., XML databases), graph data (e.g., ontology and graph databases), and now array databases are offering their benefits as well:

- A **high-level query language** allows users (typically: application developers such as data scientists) to describe the result, rather than a particular algorithm leading to this result. For example, a two-line array query typically would translate into pages of procedural code. In other words:

[1] Also NoSQL approaches, while initially denying usefulness of high-level query languages, are gradually (re-) introducing them – see MongoDB, Hive, Pig, etc.

users do not need to deal with the particularities of programming. The data center, conversely, has a safe client interface – accepting any kind of C++ or python code and running it inside the firewall is a favorite nightmare of system administrators.

- **Transparent storage management ("data independence")**. While this idea sometimes still is alien to data centers which are used to knowing the location of each byte on disk this transparency has the great advantage of (i) simplifying user access and (ii) allowing to reorganize internally without affecting users – for example, to horizontally scale a service. And, honestly: in a JPEG file, do we know the location of a particular pixel? We can operate them well without knowing these details.
- **Concurrency and access control.** Given that a large number and variety of users are querying large amounts of data it is indispensable to manage access. Avoiding inconsistencies due to parallel modifications of data is addressed by concurrency control with transaction support. Role-based access control allows adjusting access for user groups individually. Particularly with arrays, granularity of access control must go below object level for selectively managing access to arbitrary areas within datacubes, essentially performing access control down to pixel level. Also, due to the high processing load that array queries may generate it is important to enforce quota.

Array databases [9] provide flexible, scalable services on massive multi-dimensional arrays, consisting of storage management and processing functionality for multi-dimensional arrays which form a core data structure in science and engineering. They have been specifically designed to fill the gaps of the relational world when dealing with large binary datasets of structured information and have gained traction in the last years, in scientific communities as well as in industrial sectors like agriculture, mineral resource exploitation etc. 1-D sensor data, 2-D satellite and medical imagery, 3-D image timeseries, 4-D climate models are all at the core of virtually all science and engineering domains. The currently most influential array database implementations are, in historical order, rasdaman [10][12][20][20][7][38] and SciDB [16][19]; Fig. 2 gives a brief outline on the historical development of this field. Each of them allows querying the data based on the array's properties and contents using declarative languages that usually allow for a large degree of flexibility in both query formulization and internal query optimization techniques. Processing of arrays is core functionality in such databases with large sets of operations, ranging from simple sub-setting up to statistics, signal and image processing, and general Linear Algebra. A first Array Database workshop has been held in Uppsala in 2011 [4].

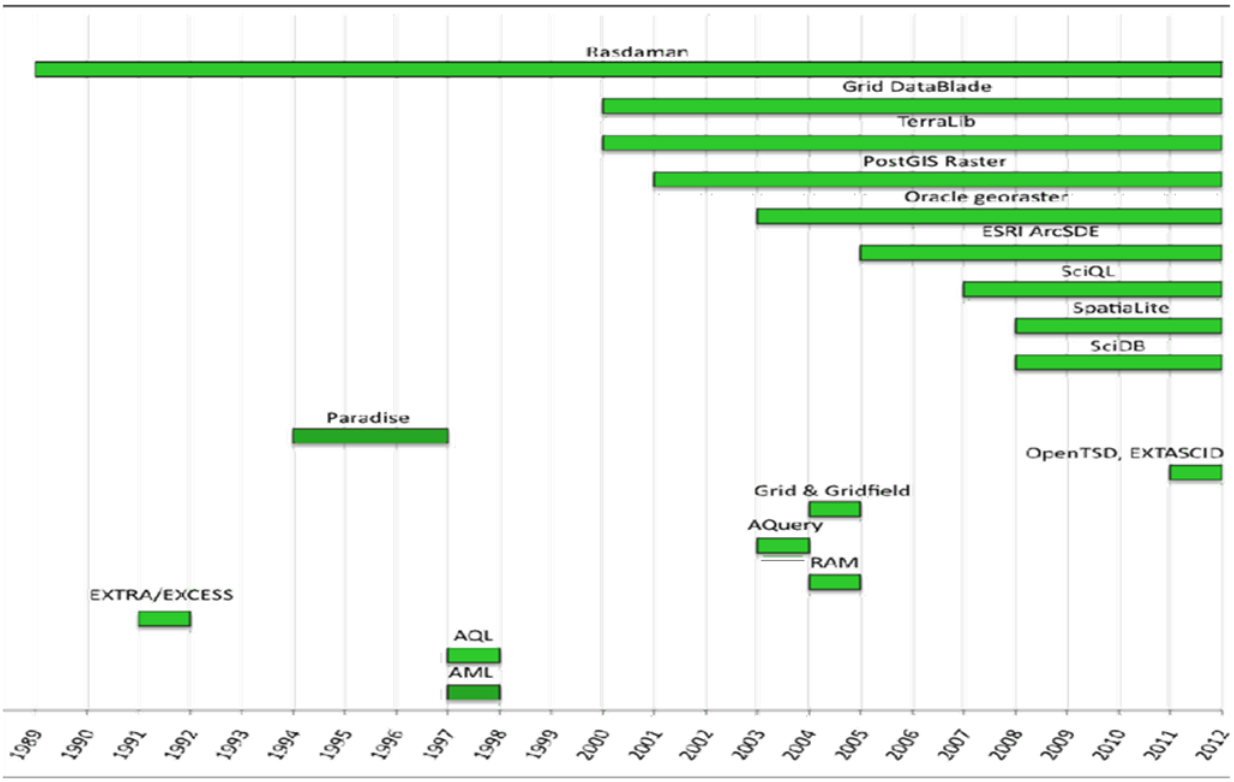

Fig. 2. Early history of Array Databases

The remainder of this report is organized as follows. In the next section we introduce to the concepts of array querying. In Section X we provide a brief discussion of open data versus open source versus open standards. An overview on Array (database) standards is given in Section X, followed by an overview of technology currently available in Section X and a slate of publicly accessible array (database) services in Section X. In Section X we provide a technical comparison of the various technologies. Section X concludes the report.

2 Array Database Concepts

This section can be skipped safely, it is helpful but not strictly necessary to understand the analysis provided lateron.

2.1 The Array Model

Formally, a d-dimensional array is a function $a: D \rightarrow V$ with a domain consisting of the d-fold Euclidean cross product of closed integer intervals:

$$D = \{lo_1, \ldots, hi_1\} \times \ldots \times \{lo_d, \ldots, hi_d\} \text{ with } lo_i \leq hi_i \text{ for } 1 \leq i \leq d$$

where V is some non-empty value set, called the array's cell type. Single elements in such an array we call cells. Arrays sometimes are popularly referred to as datacubes emphasizing the higher dimensions, from 3 onwards. Still, the concept encompasses all dimensions, including 1-D and 2-D (0-dimensional arrays constitute a border case which can be considered as a single value - it is more or less a matter of taste whether to consider them arrays or not).

This understanding is identical to mathematics where *vectors* (or *sequences*) represent 1-D arrays, *matrices* form 2-D arrays, and *tensors* represent higher-dimensional arrays.

Tomlin has established a so-called Map Algebra [43] which categorizes array operations depending on how many cells of an input array contribute to each cell of the result array; here is an excellent compressed introduction. While Map Algebra was 2-D and has been extended to 3-D lateron, AFATL Image Algebra [39] is n-dimensional by design. Array Algebra [11] has been influenced by AFATL Image Algebra when establishing a formal framework for n-D arrays suitable for a declarative query language.

2.2 Querying Arrays

Although array query languages heavily overlap there is not yet a common consensus on operations and their representation. For the brief introduction we rely on Array Algebra [11] because it is a powerful, minimal formal framework of well understood expressiveness and also forms the theoretical underpinning of the forthcoming ISO Array SQL standard, SQL/MDA (see standards section). In passing we note that array operations, being 2nd order with functions as parameters, introduce functionals. Array Algebra relies on only three core operators: An array constructor, an aggregator, and an array sort operation (which we skip for this introduction). We introduce these in turn, based on the ISO SQL/MDA syntax.

2.2.1 Deriving arrays

The `mdarray` operator creates an array of a given extent and assigns values to each cell through some expression which may contain occurrences of the cell's coordinate. Sounds complicated? Let us start simple: assume we want to obtain a subset of an array A. This subset is indicated through array coordinates, i.e., we extract a sub-array. For a d-dimensional array this subset can be defined through a d-dimensional interval given by the lower corner coordinate (lo_1, ..., lo_d) and upper corner coordinate (hi_1,...,hi_d), respectively. To create the subset array we write

```
mdarray [ x( lo1:hi1, ..., lod:hid ) ]
elements a[x]
```

This extraction, which retains the dimensionality of the cube, is called trimming. Commonly this is abbreviated as

```
a[ lo1:hi1, ..., lod:hid ]
```

We can also reduce the dimension of the result by applying slicing in one or more coordinates. This means, instead of the $lo_i:hi_i$ interval we provide only one coordinate, the slice position s_i. Notably, if we slice d times we obtain a single value (or, if you prefer, a 0-D array), written as:

```
mdarray [ x(s1, ..., sd) ]
elements a[x]
```

or in its shorthand

```
a[ s1, ..., sd ]
```

which resembles the common array cell access in programming languages. Fig. 3 shows some examples of trimming and slicing on a 3-D array.

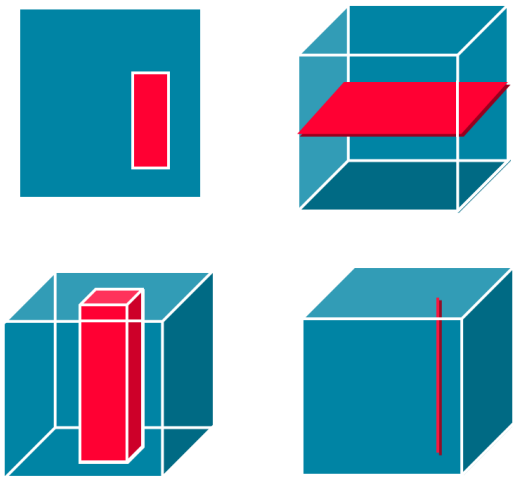

Fig. 3. Various types of subsetting from an array: trimming (left, which keeps the original dimension) and slicing (which reduces the number of dimensions, right)

Now let as assume we want to change the individual cell values rather than doing extraction, for example deriving the logarithm of some input array of given domain extent D with axes ax and y:

```
mdarray mdextent( D )
elements log( a[x,y] )
```

An example for a binary operator is addition of two images:

```
mdarray mdextent( D )
elements a[x,y] + b[x,y]
```

In fact, any unary or binary operation defined on the input arrays' cell types "induces" a corresponding array operation. For binary operations - also referred to as array joins - we require that both operand arrays share the same spatial extent so that the pairwise matching of array cells is defined. Syntactically, we abbreviate such `marray` operations so that the above example can be written as

```
a + b
```

With this simple rule we have obtained already all the well-known arithmetic, Boolean, exponential, and trigonometric operations.

Extending binary to n-ary functions we find two practically useful operations, the case and concat operators. Following the syntax of SQL we can write an array case (or "if" operator) as in the following example which performs a traffic light classification of array values, based on thresholds t1 and t2:

```
case
  when a > t2 then {255,0,0}
  when a > t1 then {0,255,255}
              else {0,255,0}
end
```

Another useful operation is array concatenation. We define, for two arrays a with domain A and b with domain B,

```
a concat b := mdarray  x in (A union B)
              elements case
                       when x in A then a[x]
                                   else b[x]
                       end
```

Obviously, the union of the input domains must be a valid array domain again. It is straightforward to extend concatenation to an n-ary function provided the input array domains altogether form a valid array partition.

2.2.2 Aggregating arrays

All the above operations have served to derive a new array from one or more given arrays. Next, we look at the *condenser* which - in analogy to SQL aggregation - allows deriving summary values. The general condenser iterates over an array covering the domain indicated and aggregates the values found; actually, each value can be given by a location-aware expression. The following example adds all cell values of a in domain D with axes x and y (which obviously must be equal to or contained in the domain of array a):

```
mdaggregate +
over mdextent( D )
using a[x,y]
```

This can be abbreviated as

```
sum(a)
```

Not all operations can act as condensers as they must be form a monoid in order for the aggregation to work. Common candidates fulfilling this criterion are `sum`, `avg`, `min`, `max`, `exists`, and `forall`.

2.2.3 Operator combinations

The operators illustrated can all be combined freely to form expressions of arbitrary complexity. We demonstrate this through two examples.

Example 1: The matrix product of a and b, yielding an result matrix of size m✕p.

```
mdarray  mdextent( i(0:m), j(0:p) )
elements mdaggregate +
         over        mdextent( k(0:n) )
         using       a [ i, k ] * b [ k, j ]
```

Example 2: A histogram over an 8-bit greyscale image.

```
mdarray  mdextent( bucket(0:255) )
elements mdcount( img = bucket )
```

This way, general operations from image / signal processing, statistics, and Linear Algebra (up to, say, the Discrete Fourier Transform) can be expressed.

2.2.4 Array integration

Some systems operate on arrays standalone, others integrate them into a host data model, typically: relations. Following ISO SQL we embed arrays into the relational model as a new column type which is shared by the majority of systems such as rasdaman, PostgreSQL, Oracle, and Teradata. This offers several practical advantages, such as a clear separation of concerns in query optimization and evaluation which eases mixed optimization [34]. For example, we can define a table of Landsat images as follows:

```
create table LandsatScenes(
    id: integer not null,
    acquired: date,
    scene: row( band1: integer, ..., band7: integer )
           mdarray [ 0:4999,0:4999]
)
```

which can be queried like this example shows:

```
select  id, encode(  (scene.band1-scene.band2)
                   / (scene.band1+scene.band2)), "image/tiff" )
```

```
from    LandsatScenes
where   acquired between "1990-06-01" and "1990-06-30" and
        mdavg(scene.band3-scene.band4)/(scene.band3+scene.band4))>0
```

A notable effect is that now data and metadata reside in the same information space and can be accessed and combined in one and the same query. Hence, in future the age-old distinction between data and metadata can be overcome.

2.3 Array Database Architectures

2.3.1 Storage

Access patterns on arrays are strongly linked to the Euclidean neighborhood of array cells (Fig. 4), therefore it must be a main goal of any storage engine to preserve proximity on persistent storage through some suitable spatial clustering. It is common, therefore, to partition n-D arrays into n-D sub-arrays called tiles [12] or chunks [40] which then form the unit of access to persistent storage.

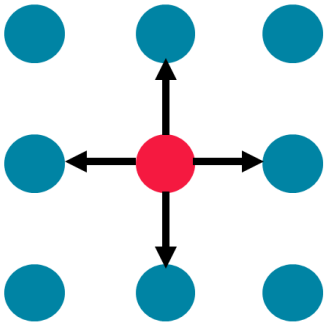

Fig. 4. n-D Euclidean neighborhood of array cells

Obviously, the concrete partitioning chosen greatly affects disk traffic and, hence, overall query performance. By adjusting the partitioning – statically in advance or dynamically at query time – to the workloads, the number of partitions fetched from persistent storage can be minimized, ideally: to a single disk access (Fig. 5). The challenge is to find a partitioning which supports a given workload. For example, when building x/y/t remote sensing data cubes imagery comes in x/y slices with a thickness of 1 along t. Time series analysis, on the contrary calls for cutouts with long time extent and (possibly) limited spatial x/y extent.

While this principle is generally accepted partitioning techniques vary to some extent. PostGIS Raster allows only 2D x/y tiles and suggests tile sizes of 100x100 pixels [37]. Teradata arrays are limited to less than 64 kB [44]. SciDB offers a two-level partitioning where smaller partitions can be gathered in cont-ainer partitions. Further, SciDB allows overlapping partitions so that queries requiring adjacent pixels (like in convolution operations) do not require reading the neighboring partitions [41]. In rasdaman, a storage layout sublanguage allows to define partitioning along several strategies [6]. For example, in “directional tiling” ratios of partition edge extents are indicated, rather than absolute sizes; this allows to balance mixed workloads containing, e.g., spatial timeslice extraction and temporal timeseries analys-

is. In the “area of interest tiling” strategy, hot spots are indicated and the system automatically determines an optimal partitioning.

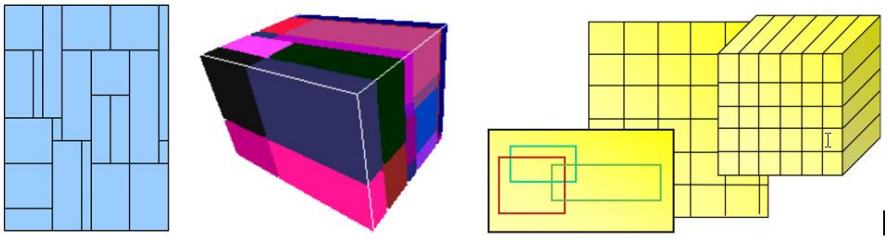

Fig. 5. Sample tiling of 2-D and 3-D arrays (left) and
rasdaman tiling strategies area-of-interest, regular, and directional (right)

To quickly determine the partitions required – a typical range query – some spatial index, such as the R-Tree, proves advantageous. As opposed to spatial (i.e., vector) databases the situation with arrays is relatively simple: the target objects, which have a box structure (as opposed to general polygons), partition a space of known extent. Hence, most spatial indexes can be expected to perform decently.

Often, compression of tiles is advantageous [20]. Still, in face of very large array databases tertiary storage may be required, such as tape robots [40][38].

2.3.2 Processing

When it comes to query evaluation it turns out that, in general, array operations are heavily CPU bound; this is contrary to relational query processing which typically is I/O bound. Some array operations are trivially parallelizable, such as cell-wise processing and combination (which Tomlin [43] calls “local” operations) and simple aggregations. These can easily be distributed both on local processing nodes like multicore CPUs and general-purpose GPUs and remote nodes, like servers in a cloud network. Others have to be carefully analyzed, transformed and sometimes even rewritten in different sets of operations to gain such parallelizable characteristics, e.g. joins on differently partitioned arrays, histogram generators and, in general, array constructors with non-serial access patterns.

The following is a non-exhaustive list of optimizations proven effective in Array DBMSs:

- **Parallelization.** The fact that array operations involve applying the same operation on a large number of values, and also the observation that tiles map naturally to CPU cores sometimes leads to the hasty conclusion that array operations per se are "embarrassingly parallel". While this holds for simple operations, such as unary induced operations like "log(a)", this is by far not true in general. Already binary operations like "a+b" pose challenges - for example, both operand arrays can reside on different nodes, even data centers, and they may have an incompatible tiling which calls for advanced methods like Array Join [5]. Additional complexity, but also opportunities, comes with Linear Algebra operations ranging from matrix multiplication over QR decomposition up to Fourier Transform and PCA, to randomly pick a few examples. Parallelization across several cores in one compute node (effectively, a shared-all architecture) allows exploiting vertical scalability; distributed processing utilizes the same principle of sharing

workloads, but across several compute nodes (shared-nothing architecture) – in case of a cloud, typically homogeneous nodes sitting close by, in the case of federations among data centers heterogeneous nodes with individual governance and higher-latency network connections. Criteria for splitting queries across multiple systems may include data location, intermediate results transfer costs, current resource availability, and several more.

- **Mixed hardware**. Compiling queries into code for CPU, GPU, FPGA, etc. can greatly speed up processing time. However, mixed hardware evaluation poses non-trivial problems which still are under active research.
- **Approximative caching**. Caching the results of final and intermediate processing steps helps significantly in case where the same or similar queries come in frequently. For example, during disasters there will be lots of queries on the disaster region, issued by mitigation forces and the general public. With arrays we encounter the particular challenge that these queries will likely not hit the exact same region, but will differ more or less on the area to be accessed. Hence, it is of advantage if the query engine can reuse also partially matching areas in arrays [31].

Generally, parallelization in Array Databases is not constrained to the rigid "Map() followed by Reduce()" pattern of Hadoop-style systems, but can look at each query individually. This opens up more opportunities, but is often nontrivial to implement. In Array Databases – as in database technology in general – two main techniques are known for finding out how to best orchestrate an incoming query based on the speedup methods available in the system:

- **Query rewriting.** This technique, which is long known in relational database query processing, looks at an incoming query whether it can be rephrased into an equivalent one (i.e., returning the same result), however, with less processing effort. To this end, the system knows a set of rewrite rules like "left hand side expression returns same result as right hand side, but we know right-hand side is faster". Where do these rules come from? Actually, this is a nice example for the usefulness of a formal semantics of a language; Relational and Array Algebra naturally lead to algebraic equivalences which can be directly written into code. In the case of rasdaman, there are about 150 such rules currently.
  The following example (Fig. 6) illustrates the principle, with a rule saying "adding two images pixelwise, and then computing the average value, is equivalent to first computing the averages individually, and then add the result". In the first case, array tiles have to be streamed three times in the server whereas in the second case there are only two tile streams – the final addition is over two scalars, hence negligible in cost. Bottom line, replacing an occurrence of the left-hand side pattern by the right-hand side pattern saves 1/3 of the computing effort.

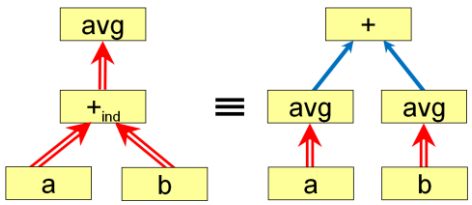

Fig. 6. Sample equivalence rule for array query rewriting: "$avg(a+b) \equiv avg(a)+avg(b)$"

- **Cost-based optimization** attempts to find an efficient execution plan out of the - usually large - search space of possible plans for a given query. In contrast to query rewriting, this involves knowledge (i.e.: estimation) of the costs of processing. Parameters influencing costs include the number of tiles to be read from disk, location of tiles in case of a distributed system, the number and complexity of operations, and several more.

Note that reformulating and compiling queries is not a time consuming task. Experience with rasdaman shows that the optimization steps altogether take about a millisecond.

2.4 Client interfacing

While "datacubes" represent a convenient logical view on massive multi-dimensional data this does not mean clients need to see data in such a shape. Very often, clients will do some extraction and aggregation, thereby reducing and changing dimensionality away from the original. More importantly even, users should be able to remain as much as possible within their comfort zone of well known tools - for example, simple map navigation should still be able through clients like OpenLayers and Leaflet, embedding into Web GIS should support tools like QGIS and ESRI ArcGIS, virtual globes like NASA WebWorldWind and Cesium should be supported, whereas high-end analytics calls for access to datacubes through R and python.

2.5 Related Technology

Array databases, by definition, are characterized by offering a declarative query language on n-D arrays. Such technology can e implemented in various ways - as will be demonstrated by the systems overview in the next section - each coming with its individual characteristics. However, we will also look beyond the pure Array Database category and give a glance at other array technology, including

- array engines offering only procedural interfaces (rather than a query language), often implemented in some scripting language (e.g., python), rather than running directly compiled machine code (e.g., C++). Typically, these are constrained in functionality as users can only invoke the functions provided, but cannot compose them to larger tasks – hence, they lack the flexibility of databases.
- Command-line tools which form possible components of array services, but do not constitute a complete service tool per se. Typically, these are useful for services inside a data center where data experts at the same time are experienced full-stack developers.
- Libraries that provide array functionality, but do not constitute a server and do not have a query concept (but rather a procedural API).

This way, we aim at providing a context for the novel category of Array Databases.

3 Open Data, Open Source, Open Standards

In the era where openness has become highly valued we sometimes observe confusion about the meaning and consequence of an "open X". We, therefore, briefly discuss three core terms heavily debated in the science data domain. Note that the goal is not to define or even explain in detail - this has been done many times elsewhere already -, but to specifically relate these three terms to each other.

3.1 Open Data

This addresses accessibility of data. Data are said to be open if they can be accessed without any restriction stemming from constrained user groups, etc. A related term is "Free Data" meaning that access is free of cost.

An indirect obstacle to free access, aside and independently from organizational restrictions, can be the difficulty of access due to reasons such as uncommon data formats, unwieldy data granules (such as 100 GB TIFF files), access interfaces requiring high technical backgrounds, or interfaces posing particular hardware requirements (high client-side hardware resource needs, high-bandwidth connection, etc). Hence, offering open data also has an implication on the ease of use of the data offered. In this context, an interesting and widely embraced initiative has been launched by the USGS Landsat team coining the term Analysis Ready Data. In this approach, data centers tentatively undertake high effort in preparing (homogenizing, restructuring, cleansing, etc.) data in a way that reduces such intrinsic obstacles to data access.

3.2 Open Source

This term refers to the software used, e.g., to serve or access data (i.e., servers and clients in Web-based information systems). By way of background, most programs are written by human developers in som ehigh-level language which is closer to human perception concepts than the computer's machine language - hence, programming becomes more efficient, less error prone, and resulting programs are better to maintain. For each language there are special programs - called compilers or interpreters - translating this "souce code" into "object code" which can be executed by a particular CPU. Note that for one and the same language different compilers may exist, and do so in practice - we will need this fact later.

Obviously, the machine code is hard to understand for humans, as opposed to the high-level source code which is digestible at least by programming experts. Hence, source code allows to find out what a program really does - whether it does the right thing, does computations correctly and without flaws like undue rounding errors, does not contain malicious code, etc. Of course, detecting any such issue requires high effort by skilled programmers, so not everybody is able to benefit from the openness of the source code.

Further, even open source code runs in the particular "environment" of the computer hosting the service. As such, the program will use external libraries whose source code may or may not be open, and it has been derived from the source code through a compiler which itself may or may not be open.

Hence, even when inspected by experts openness of the source code of the tool under consideration is not necessarily a guarantee for completely overseeing its effects.

In particular, for data scientists (i.e., not computer scientists) it is generally not possible to verify the validity of open source code - and be it just for the lack of time to inspect all source code involved.

Generally, speaking, both open source and proprietary software build and maintenance approaches have their individual advantages and disadvantages. In today's planetary software ecosystem we find a wide spectrum of business models, ranging from fully open source over mixed models (like dual license) to fully closed, proprietary software (such as license sales or leases) - and often we find them in combination (such as running open-source clients on closed-source MS-Windows).

3.3 Open Standards

In Information Technology, standards typically establish data formats and interfaces between software components so that software artifacts written by different, independent producers (say, different companies or different departments within a company) still can communicate and perform a given task jointly. Building software based on only interface knowledge and without knowledge about the internals of how a component establishes the behaviour described by the interface definition is a key achievement in Software Engineering; without such boxed thinking, the complexity of today's software would be absolutely intractable and unmanageable.

Like with data, a standard is called open if it is accessible to everybody without discriminating; some of those standards additionally are free of cost (such as with the Open Geospatial Consortium, OGC) while others are available against a (usually moderate) fee (such as with ISO).

Some standardization bodies offer compliance suites which allows validating implementations against the standard. One example is the extensive OGC compliance test suite.

Importantly, it is sufficient for some tool to know its interface specifications ("if I input X I will get Y"). If this specification is an open standard, and if the tool has been confirmed to pass the corresponding compliance test, then the behaiour of this tool can be trusted with respect to this standard (of course, there may be further unwanted behaviour not addressed by the compliance test - for example, such a test will typically concentrate on functionality, but not on security).

Examples are mainfold: we trust SQL query language implementation, regardless whether the database management system is open or closed source; we trust our C++ compilers, python engines, numerical libraries, operating systems, etc. - at least concerning the core question addressed here: does this code provide me with the true, valid data (read from disk or processed)? And, for that matter, we trust the underlying hardware which ultimately executes the code.

3.4 Conclusion

Concluding, open data and open source and open standards are three different concepts, each one addressing separate concerns in the first place. Open data access is desirable from many aspects, although there are valid reasons for some data to be not openly accessible. The service software in

particular plays an instrumental role in guaranteeing the promise of open data. Open source as such, though, is not a guarantee (and not a required prerequisite) for open data - open standards serve a much better role in this, although with the caveat that standards do not make a statement about the complete client or server stack, but only about the particular aspect addressed by the standard. However, by using well-crafted standards (ideally coming with a solid mathematical underpinning), such as the ISO SQL relational query language or the OGC WCPS geo datacube query language, a substantial contribution towards the Holy Grail of open data can be made. The interoperability established thereby - in this context: different servers using identical data will deliver identical results - constitutes a major advantage whose benefits are by far not leveraged in full today.

4 Array Standards

Several standards relevant for large-scale array processing are already in place or under development. Such standards may be both domain independent (such as ISO Array SQL) or domain specific (such as the OGC WCPS geo raster query language). For each standard, its adoption status is provided.

4.1 Domain Neutral

- Array SQL (data and processing standard)
  - Full title: ISO 9075 SQL Part 15: Multi-Dimensional Arrays (MDA)
  - Issuing Body: ISO (SC 32 / WG 3)
  - Description: SQL extension with domain-neutral definition and queries on massive multi-dimensional arrays ("datacubes").
  - Adoption Status: DIS ballot started in August 2017
  - Further information:
    - D. Misev, P. Baumann: Homogenizing Data and Metadata Retrieval in Scientific Applications. Proc. ACM CIKM DOLAP, Melbourne, Australia, October 23, 2015, pp. 25 - 34

4.2 Earth & Planetary Sciences

- OGC Coverages (geo datacube data standard)
  - Full title: Coverage Implementation Schema (CIS) [formerly also known as: GMLCOV]
  - Issuing Body: Open Geospatial Consortium (OGC)
  - Description: service-independent data model for spatio-temporal regular and irregular grids, point clouds, and general meshes. As opposed to ISO 19123 (see below), this is concrete enough to be interoperable and conformance testable down to pixel level.
  - Adoption Status:
    - adopted by OGC
    - under adoption by ISO as DIS 19123-2
    - adopted by EU INSPIRE, with slight modifications (re-harmonization with OGC coverage standard under work)
  - Further information:
    - authoritative standards page: http://www.opengeospatial.org/standards/wcs
    - tutorials and webinars: http://earthserver.eu/webinars
    - P. Baumann, E. Hirschorn, J. Maso, A. Dumitru, V. Merticariu: Taming Twisted Cubes. Proc. ACM SIGMOD Workshop on Managing and Mining Enriched Geo-Spatial Data (GeoRich), San Francisco, USA, June 26 - July 01, 2016
    - P. Baumann: Beyond Rasters: Introducing The New OGC Web Coverage Service 2.0. Proc. ACM SIGSPATIAL GIS, San Jose, USA, November 2-5, 2010
- ISO Coverages (geo datacube data standard)
  - Full title: OGC Abstract Topic 6 (identical to ISO 19123)
  - Issuing Body: ISO (TC211)

    - Description: abstract, generic data model for spatio-temporal coverages. This forms the conceptual basis for the Coverage Implementation Schema (CIS) under adoption by ISO (See above).
    - Adoption Status:
        - adopted since 2004 as 19123, under revision by ISO to become 19123-1
    - Further information:
        - None currently
- OGC WCS (geo datacube access and processing standard)
    - Full title: Web Coverage Service
    - Issuing Body: Open Geospatial Consortium (OGC)
    - Description: modular Web service for accessing spatio-temporal regular and irregular grids, point clouds, and general meshes
    - Adoption Status:
    - adopted OGC standard since 2012
        - adopted by EU INSPIRE in December 2016
        - planned for adoption by ISO
    - Further information:
        - authoritative standards page: http://www.opengeospatial.org/standards/wcs
        - tutorials and webinars: http://earthserver.eu/webinars
        - P. Baumann: Beyond Rasters: Introducing The New OGC Web Coverage Service 2.0. Proc. ACM SIGSPATIAL GIS, San Jose, USA, November 2-5, 2010
- OGC WCPS (processing standard)
    - Full title: Web Coverage Processing Service
    - Issuing Body: Open Geospatial Consortium (OGC)
    - Description: geo raster query language for massive spatio-temporal datacubes over regular or irregular grids
    - Adoption Status:
        - adopted OGC standard since 2009
        - optional part in INSPIRE Coverage Download Services
    - Further information:
        - authoritative standards page: http://www.opengeospatial.org/standards/wcps
        - tutorials and webinars: http://earthserver.eu/webinars
        - P. Baumann: The OGC Web Coverage Processing Service (WCPS) Standard. Geoinformatica, 14(4)2010, pp 447-479

5 Array Technology

This page collects technology for handling massive multi-dimensional arrays. While emphasis is on Array Databases, other technologies addressing arrays are mentioned as well as long as substantial array support can be evidenced. Please observe Etiquette (see bottom).

Array databases naturally can do the "heavy lifting" in multi-dimensional access and processing, but arrays in practice never come alone; rather, they are ornamented with application-specific metadata that are critical for understanding of the array data and for querying them appropriately. For example, in geo datacubes querying is done typically on geographical coordinates, such as latitude and longitude; the system needs to be able to translate queries in geo coordinates into the native Cartesian index coordinates o arrays. In all applications using timeseries, users will want to utilize date formats - such as ISO 8601 supporting syntax like "2018-02-20" - rather than index counting. For cell types, it is not sufficient to just know about integer versus floating-point numbers, but it is important to know about units of measure, null values (note that sensor data do not just deliver one null value, such as traditional database support, but multiple null values with individual semantics).

Coupling array queries with metadata query capabilities, therefore, is of high practical importance; ISO SQL/MDA, with its integration of arrays into the rich existing framework of the SQL language, shows one possible way. If that appears too complex to implement, silo solutions with datacube support are established. Specifically in the Earth science domain an explosion of domain-specific "datacube" solutions can be observed recently (see, e.g., the EGU 2018 datacube session), usually implemented in python using existing array libraries. We, therefore, also look at domain-specific "datacube" tools as well.

This state of the art review on array service implementations is organised as follows. First, Array Databases are inspected which offer generic query and architectural support for n-D arrays. Next, known object-relational emulations of arrays are listed. MapReduce-type systems follow as a substantially different category of data systems, which however often is mentioned in the context of Bi Data. After that, systems are listed which do not fall into any of the above categoris. Finally, we list libraries (as opposed to the aforementioned complete engines) and n-D array data formats.

5.1 Array Database Systems and Related Technologies

5.1.1 Systematics

This section inspects Array Database and related technology. As recently a significant boom in array systems can be observed that an increasing number of technologies is being announced, at highly varying stages of maturity. Thanks to the blooming research and development it can be expected that further systems emerge soon which have not found their way into this report. The landscape of systems encountered has been grouped into the following categories (see also Section 2.5):

- **Array Database systems** characterized by a query language, multi-user operation, storage management, and access control mechanisms. These can be subdivided into
    - **Full-stack Array Databases** which are implemented from scratch (ex: rasdaman, SciDB)

    - **Add-ons to existing database systems** which are implemented as extra layers to existing DBMSs (ex: EXTASCID), as object-relational extensions (ex: PostGIS Raster, Teradata Arrays, OracleGeoRaster), or through direct DBMS kernel coding (ex: SciQL).
- **Array tools** encompassing **command-line oriented** and **libraries** that provide array functionality, but do not constitute a server; the central distinguishing criteria are that (i) they do not offer a query concept, but rather a procedural API (where each call can accomplish just one piece of functionality, as opposed to arbitrarily complex user queries in databases), and (ii) they do not accept queries via Internet, but rather require being logged in on the server machine for executing shell commands (ex: Ophidia) or writing own embedding code in some scripting language like python (ex: Wendelin.core, xarray, TensorFlow) or a compiled language like C++ (ex: boost::geometry, xtensor). Such approaches appear useful inside a data center where data experts at the same time are experienced full-stack developers, as opposed to data scientists who generally prefer high-level languages like R.
  As such, these tools and libraries form possible components of array services, but do not constitute a complete service tool per se.
- **MapReduce type array engines** allowing multi-dimensional array processing based on top of Hadoop or Spark.

5.1.1 Array DBMSs - Full-Stack

In this category we find database systems with the characteristic service features – a query language, multi-user operation, etc.

5.1.1.1 rasdaman ("raster data manager")

Description: Rasdaman has pioneered the field of Array Databases, with publications since 1992. This array engine allows declarative querying of massive multi-dimensional arrays, including distributed array joins. Server-side processing relies on effective optimization, parallelization, and use of heterogeneous hardware for retrieval, extraction, aggregation, and fusion on distributed arrays. The architecture resembles a parallelizing peer federation without a single point of failure. Arrays can be stored in the optimized rasdaman array store or in standard databases; further, rasdaman can operate directly on any pre-existing archive structure. Single rasdaman databases exceed a PB [3], and queries have been split successfully across more than 1,000 cloud nodes [21]. The rasdaman technology has coined the research field of Array Databases [10]and is blueprint for several Big Data standards, such as the ISO SQL/MDA (Multi-Dimensional Arrays) candidate standard [34] and the OGC Web Coverage Service (WCS) "Big Geo Data" suite with its geo datacube query language, Web Coverage Processing Service (WCPS) [13].

Source code: www.rasdaman.org/Download for the open-source *rasdaman community* edition (LGPL for client libraries, GPL for server – so can be embedded in commercial applications); for the proprietary *rasdaman enterprise* edition see www.rasdaman.com.

Public demo site and further information:

- http://standards.rasdaman.com

Publications (excerpt only - see full list):

- ISO FDIS 9075 SQL Part 15: Multi-Dimensional Arrays
- P. Baumann: Array Databases and Raster Data Management. In: T. Özsu, L. Liu (eds.): Encyclopedia of Database Systems, Springer, 2017
- D. Misev, P. Baumann: The Open-Source rasdaman Array DBMS. VLDB Big Data Open Source Systems (BOSS) Workshop, New Delhi, India, September 09, 2016
- P. Baumann, V. Merticariu: On the Efficient Evaluation of Array Joins. Proc. IEEE Big Data Workshop Big Data in the Geo Sciences, Santa Clara, US, October 29, 2015
- P. Baumann: On the Management of Multidimensional Discrete Data. VLDB Journal 4(3)1994, Special Issue on Spatial Database Systems, pp. 401 - 444
- P. Baumann: A Database Array Algebra for Spatio-Temporal Data and Beyond. Proc. Intl. Workshop on Next Generation Information Technologies and Systems (NGITS '99), July 5-7, 1999, Zikhron Yaakov, Israel, Springer LNCS 1649
- Peter Baumann: Language Support for Raster Image Manipulation in Databases. Proc. Int. Workshop on Graphics Modeling, Visualization in Science & Technology, Darmstadt/Germany, April 13 - 14, 1992

5.1.1.2 SciDB

Description: SciDB is an Array DBMS following the tradition of rasdaman. SciDB employs its own query interface offering two languages, AQL (Array Query Language) and AFL (Array Functional Language). Its architecture is based on a modified Postgres kernel in the center plus UDFs (User-Defined Functions) implementing array functionality, and also effecting parallelization.

Website: https://www.paradigm4.com/

Source code: https://drive.google.com/drive/folders/0BzNaZtoQsmy2aGNoaV9Kdk5YZEE (last version of source code; more recent SciDB versions – current at the time of this writing is 18.1 – do not publish the source code any longer)

(dual license model, see details; community version is Affero: not allowed for commercial purposes)

5.1.1.3 SciQL

Description: SciQL was a case study extending the column-store DBMS MonetDB with array-specific operators. As such, n-D arrays were sequentialized internally to column-store tables (i.e., there is no dedicated storage and processing engine).

Website: https://projects.cwi.nl/scilens/content/platform.html

Source code: (could not find it - not with MonetDB)

5.1.1.4 EXTASCID

Description: EXTASCID is a complete and extensible system for scientific data processing. It supports natively both arrays as well as relational data. Complex processing is handled by a metaoperator that can execute any user code. EXTASCID is built around the massively parallel GLADE architecture for data aggregation. While it inherits the extensibility provided by the original GLA interface implemented in

GLADE, EXTASCID enhances this interface considerably with functions specific to scientific processing. (source).

Website: http://faculty.ucmerced.edu/frusu/Projects/GLADE/extascid.html

Source code: (could not find it – likely not publicly available)

5.1.2 Array DBMSs – Object-Relational Extensions

Object-relational capabilities in relational DBMSs allow users (usually: administrators) to define new data types as well as new operators. Such data types can be used for column definitions, and the corresponding operators can be used in queries. While this approach has been implemented by several systems (see below) it encounters two main shortcomings:

- An array is not a data type, but a data type *constructor* (sometimes called "template"). An instructive example is a stack: likewise, it is not a data type but a template which needs to be instantiated with some element data type to form a concrete data type itself - for example, by instantiating `Stack<T>` with `String` - often denoted as `Stack<String>` - one particular data type is obtained; `Stack <Integer>` would be another one. An array template is parametrized with an n-dimensional extent as well as some cell ("pixe", "voxel") data type; following the previously introduced syntax this might be written as `Array<Extent,CellType>`. Hence, object-relational systems cannot provide the array abstraction as such, but only instantiated data types like
  `Array<[0:1023,0:767],int>`
  or
  `Array <[0:1023,0:767],struct{int red, green, blue;}>`.
  Further, as the SQL syntax as such cannot be extended such array support needs to introduce some separate array expression language. Generic array types like the rasdaman n-D array constructor become difficult at best. Further, this approach typically implies particular implementation restrictions.
- Due to the genericity of such object-relational mechanisms there is no dedicated internal support for storage management (in particular: for efficient spatial clustering, but also for array sizes), indexing, and query optimization.

Still, some systems have implemented array support in an object-relational manner as it is substantially less implementation effort than implementing the full stack of an Array DBMS, with each component crafted specifically for arrays.

5.1.2.1 PostGIS Raster

Description: "Raster" is a PostGIS type for storing and analyzing geo raster data. Like PostGIS in general, it is implemented using the extension capabilities of the PostgreSQL object-relational DBMS. Internally, raster processing relies heavily on GDAL. Currently, PostGIS Raster supports x/y 2D and, for x/y/spectral, 3D rasters. It allows raster expressions, however, not integrated with the PostgreSQL query language but passed to a raster object as strings written in a separate Map Algebra language. Large objects have to be partitioned by the user and distributed over tuples in a table's raster column; queries have to be

written in a way that they achieve a proper recombination of larger rasters from the partitions stored in one tuple each. A recommended partition size is 100x100 pixels.

Website: http://postgis.net/docs/manual-2.1/RT_reference.html

Source code: https://trac.osgeo.org/postgis/wiki/DevWikiMain

5.1.2.2 Oracle GeoRaster

Description: GeoRaster is a feature of Oracle Spatial that lets you store, index, query, analyze, and deliver raster image and gridded data and its associated metadata. GeoRaster provides Oracle spatial data types and an object-relational schema. You can use these data types and schema objects to store multidimensional grid layers and digital images that can be referenced to positions on the Earth's surface or in a local coordinate system. If the data is georeferenced, you can find the location on Earth for a cell in an image; or given a location on Earth, you can find the cell in an image associated with that location. There is no particular raster query language underneath, nor a specific array-centric architecture.

Website: http://docs.oracle.com/cd/B19306_01/appdev.102/b14254/geor_intro.htm

Source code: n.a. (closed source, proprietary)

5.1.2.3 Teradata Arrays

Description: Teradata recently has added arrays as a datatype, also following an object-relational approach. There are some fundamental operations such as subsetting; however, overall the operator do not resemble the expressive power of genuine Array DBMSs. Further, arrays are mapped to 64 kB blobs so that the overall size of a single array (considering the array metadata stored in each blob) seems to be around 40 kB. Further, there are severe restrictions: You can update only one element of the array at a time; it us unclear whether array joins are supported.

Website: https://developer.teradata.com/database/reference/array-data-type-scenario, http://info.teradata.com/htmlpubs/DB_TTU_16_00/index.html#page/SQL_Reference%2FB035-1145-160K%2Fxbk1472240940805.html%23

Source code: n.a. (closed source, proprietary)

5.1.3 Array Tools

5.1.3.1 OPeNDAP

Description: OPeNDAP ("Open-source Project for a Network Data Access Protocol") is a data transport architecture and protocol for earth scientists. OPeNDAP includes standards for encapsulating structured data, annotating the data with attributes and adding semantics that describe the data. An OPeNDAP client sends requests to an OPeNDAP server, and receives various types of documents or binary data as a response. (Wikipedia)

An array is one-dimensional; multidimensional Arrays are defined as arrays of arrays. An array's member variable MAY be of any DAP data type. Array indexes MUST start at zero. A constraint expression pro-

vides a way for DAP client programs to request certain variables, or parts of certain variables, from a data source. A constraint expression may also use functions executed by the server. See this source for details.

Website: http://www.opendap.org/

Source code: http://www.opendap.org/software/hyrax-data-server (Hyrax)

5.1.3.2 xarray

Description: xarray (formerly xray) is an open source project and Python package that aims to bring the labeled data power of pandas to the physical sciences, by providing N-dimensional variants of the core pandas data structures.Goal is to provide a pandas-like and pandas-compatible toolkit for analytics on multi-dimensional arrays, rather than the tabular data for which pandas excels. The approach adopts the Common Data Model for self- describing scientific data in widespread use in the Earth sciences: xarray.Dataset is an in-memory representation of a netCDF file. [source: xarray.pydata.org/en/stable/]

Website: http://xarray.pydata.org

Source code: http://xarray.pydata.org/en/stable/installing.html#instructions

5.1.3.3 TensorFlow

Description: TensorFlow is a tool for machine learning. While it contains a wide range of functionality, TensorFlow is mainly designed for deep neural network models.

Website: https://www.tensorflow.org/

Source code: https://www.tensorflow.org/install/

5.1.3.4 wendelin.core

Description: Wendelin.core allows you to work with arrays bigger than RAM and local disk. Bigarrays are persisted to storage, and can be changed in transactional manner. In other words bigarrays are something like numpy.memmap for numpy.ndarray and OS files, but support transactions and files bigger than disk. The whole bigarray cannot generally be used as a drop-in replacement for numpy arrays, but bigarray slices are real ndarrays and can be used everywhere ndarray can be used, including in C / python / Fortran code. Slice size is limited by virtual address-space size, which is about max 127TB on Linux / amd64. (source)

Website: https://lab.nexedi.com/nexedi/wendelin.core

Source code: https://lab.nexedi.com/nexedi/wendelin.core

5.1.3.5 Google Earth Engine

Description: Google Earth Engine builds on the tradition of Grid systems with files, there is no datacube paradigm. Based on a functional programming language, users can submit code which is executed transparently in Google's own distributed environment, with a worldwide private network. Parallelization is straightforward. After discussion of the developers with the rasdaman team, Google has added a declar-

ative “Map Algebra” interface in addition which resembles a subset of the rasdaman query language. In a face-to-face conversation at the "Big Data from Space" conference 2016, the EarthEngine Chief Architect explained that EarthEngine is relying on Google’s massive hardware rather than on algorithmic elaboration. At the heart is a functional programming language which does not offer model-based array primitives like rasdaman, nor comparable optimization.

Website: https://earthengine.google.com/

Source code: n.a., closed-source, proprietary system

5.1.3.6 OpenDataDatacube

Description: The Open Data Cube (ODC) initiative seeks to increase the value and impact of global Earth observation satellite data by providing an open and freely accessible exploitation architecture. (source). A python API specification can be found at http://datacube-core.readthedocs.io/en/stable/dev/api.html, a Web interface specification could not be found.

Website: https://www.opendatacube.org/

Source code: https://github.com/ceos-seo/data_cube_ui/blob/master/docs/datacube_install.md

5.1.3.7 xtensor

Description: xtensor is a C++ library meant for numerical analysis with multi-dimensional array expressions. xtensor provides an extensible expression system enabling lazy broadcasting, an API following the idioms of the C++ standard library, and tools to manipulate array expressions and build upon xtensor. Containers of xtensor are inspired by NumPy, the Python array programming library. Adaptors for existing data structures to be plugged into our expression system can easily be written. In fact, xtensor can be used to process numpy data structures inplace using Python’s buffer protocol. For more details on the numpy bindings, check out the xtensor-python project. (source)

Website: http://quantstack.net/xtensor

Source code: https://github.com/QuantStack/xtensor

5.1.3.8 boost::geometry

Description: Boost.Geometry (aka Generic Geometry Library, GGL), part of collection of the Boost C++ Libraries, defines concepts, primitives and algorithms for solving geometry problems. Boost.MultiArray provides a generic N-dimensional array concept definition and common implementations of that interface.

Website: http://www.boost.org/doc/libs/1_66_0/libs/multi_array/doc/index.html

Source code: https://github.com/boostorg/boost

5.1.3.9 Ophidia

Description: The Ophidia framework provides a full software stack for data analytics and management of big scientific datasets exploiting a hierarchically distributed storage along with parallel, in-memory

computation techniques and a server-side approach. The Ophidia data model implements the data cube abstraction to support the processing of multi-dimensional (array-based) data. A wide set of operators provides functionalities to run data analytics and metadata management: e.g. data sub-setting, reduction, statistical analysis, mathematical computations, and much more. So far about 50 operators are provided in the current release, jointly with about 100 primitives covering a large set of array-based functions. The framework provides support for executing workflows with various sizes and complexities, and an end-user terminal, i.e.: command-line interface. A programmatic Python interface is also available for developers.

Website: http://ophidia.cmcc.it/

Source code: https://github.com/OphidiaBigData (GPLv3)

5.1.3.10 TileDB

Description: The TileDB library manages data that can be represented as dense or sparse arrays. It can support any number of dimensions and store in each array element any number of attributes of various data types. It offers compression, high IO performance on multiple data persistence backends, and easy integration with ecosystems used by today's data scientists.

Website: https://tiledb.io/

Source code: https://github.com/TileDB-Inc/

5.1.4 MapReduce-Type Systems

5.1.4.1 Overview

MapReduce offers a general parallel programming paradigm which is based on two user-implemented functions, Map() and Reduce(). While Map() performs filtering and sorting, Reduce() acts as an aggregator. Both functions are instantiated multiple time for massive parallelization; the MapReduce engine manages the process instances as well as their communication.

Implementations of the MapReduce paradigm - such as Hadoop, Spark, and Flink - typically use Java or Scala for the Map() and Reduce() coding. While these languages offer array primitives for processing multi-dimensional arrays locally within a Map() and Reduce() incarnation here is no particular support for arrays exceeding local server main memory; in particular, the MapReduce engines are not aware of the spatial n-dimensional proximity of array partitions. Hence, the common MapReduce optimizations cannot exploit the array semantics. Essentially, MapReduce is particularly well suited for unstructured data like sets: "Since it was not originally designed to leverage the structure its performance is suboptimal" [1].

That said attempts have been made to implement partitioned array management and processing on top of MapReduce. Below some major approaches are listed.

5.1.4.2 SciHadoop

Description: SciHadoop is a Hadoop plugin allowing scientists to specify logical queries over array-based data models. SciHadoop executes queries as map/reduce programs defined over the logical data model. A SciHadoop prototype has been implemented for NetCDF data sets.

Website: DAMASC research group.

Source code: https://github.com/four2five/SciHadoop

5.1.4.3 SciSpark

Description: SciSpark is a NASA's Advance Information Systems Technology (AIST) program funded project that seeks to provide a scalable system for interactive model evaluation and for the rapid development of climate metrics and analysis to address the pain points in the current model evaluation process. SciSpark directly leverages the Apache Spark technology and its notion of Resilient Distributed Datasets (RDDs). SciSpark is implemented in a Java and Scala Spark environment.

Website: https://scispark.jpl.nasa.gov/

Source code: https://github.com/SciSpark

5.1.4.4 GeoTrellis

Description: GeoTrellis is a geographic data processing engine for high performance applications. GeoTrellis provides data types for working with rasters in the Scala language, as well as fast reading and writing of these data types to disk.

Website: http://geotrellis.io/

Source code: https://github.com/geotrellis

5.1.4.5 MrGeo

Description: MrGeo (pronounced "Mister Geo") is an open source geospatial toolkit designed to provide raster-based geospatial processing capabilities performed at scale. MrGeo enables global geospatial big data image processing and analytics. MrGeo is built upon the Apache Spark distributed processing framework.

Website: https://github.com/ngageoint/mrgeo/wiki

Source code: https://github.com/ngageoint/mrgeo

6 Publicly Accessible Array Services

Below, a selection of publicly accessible services (in RDA terminology: adopters) is listed which use Array Database technology. To be noted is the variability of the portal frontends and clients used, all uniformly mapping to Array Database technology underneath.

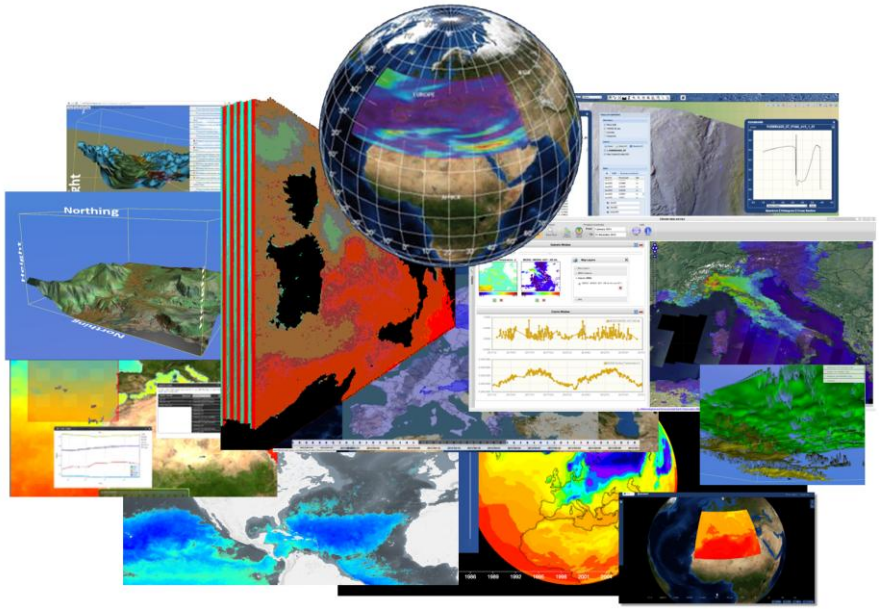

Fig. 7. Impressions of various services powered by an Aray Database system (source: rasdaman / EarthServer).

- standards.rasdaman.com (rasdaman): sample geo-oriented array use cases on 1-D through 5-D data sets. Purpose of this service is to illustrate practical use of the OGC Big Geo Datacube standards, WCS and WCPS.
- ESA Earth Observation Data Service (rasdaman): this ESA service, maintained by MEEO, currently (beginning 2018) is offering in excess of 2.5 Petabyte of Atmosphere, Land and Ocean EO products coming from the Sentinel family. A private cloud infrastructure is being set up to implement advanced access processing services on Big Data.
- ECMWF Climate Science Data Service (rasdaman): The experimental service provides access to ERA-interim reanalysis datathrough the OGC standard data access protocols WCS and WCPS. A connection to ECMWF's Meteorological Archival and Retrieval System (MARS) has been demonstrated.
- Marine Science Data Service (rasdaman): this service, offered by Plymouth Marine Laboratory (PML, UK) provides access and processing on satellite imagery for ocean colour analysis. Current offering is 70+ TB.
- PlanetServer (rasdaman): Planetary Data Service, hosted by Jacobs University, is offering geology data currently for Mars, Moon, and Vesta. Total data size is 20+ TB, based on OGC WCS and WCPS standard based interfaces.

- CODE-DE is the German Sentinel hub providing data collected by the ESA Sentinel Earth Observation satellite family. The batch-oriented Hadoop-based service of CODE-DE is currently being enhanced with interactive spatio-temporal datacube analytics using rasdaman.
- National Computational Infrastructure (NCI) Australia has an experimental service on Landsat8 data covering Australia, running rasdaman.

7 Array Systems Assessment

7.1 Systematics

We look at the systems from the perspectives

- **Functionality:** What functionality does the system offer? Are there any known restrictions?
- **Architecture:** This mainly addresses the architectural paradigms used. As such, this is not a quality criterion, but provided as background information.
- **Performance:** How fast and scalable is the tool in comparison?

This section relies on [33] and other work undertaken in this context.

Each of the criteria applied is explained first; after that, a feature matrix is presented summarizing all facts synoptically. In addition, literature is cited where the information has been harvested from. This allows recapitulating the matrix. Notably, several systems today are capable of integrating external code (e.g., SciDB, rasdaman). Therefore, it is indispensable for each functionality feature to clearly state if it is an integral part implemented in the core engine or not.

Some systems mentioned could not be considered due to resource limitations, but they are considered sufficiently similar to the ones inspected below. Examples include MrGeo and GeoTrellis as specialized Hadoop implementations offering array support.

7.2 Functional Comparison

7.2.1 Criteria

This is functionality the user (i.e., query writer) has available in terms of the data and service model. In this spirit, we also list export/import interfaces as well as known client interfaces although they do not belong to the logical level in a classic sense. Parameters investigated are the following:

Data model expressiveness:

- **number of dimensions:** what number of dimensions can an array have? Today, 3-D x/y/t image timeseries and x/y/z voxel cubes are prominent, but also 4-D x/y/z/t gas and fluid simulations, such as atmospheric weather predictions. However, other dimensions occur as well: 1-D and 2-D data appear not only standalone (as sensor and image data, resp.), but also as extraction results from any-dimensional datacubes (such as a pixel's history or image time slices). Also, higher dimensions occur regularly. Climate modellers like to think in 5-D cubes (with a second time axis), and statistical datacubes can have a dozen dimensions. Any array engine should offer support for this spectrum of dimensions.
- **extensibility** of extent along dimensions: can an existing array be extended along each dimension's lower and upper bound? Imagine a map has been defined for a country, and now is to be extended to cover the whole continent. This means: every axis must be extensible, and it must be so on both its lower and upper bounds.

- **cell data types:** support for numeric data types, for composite cells (e.g., red/green/blue pixels), etc. While radar imagery consists of single values (complex numbers), satellite images may have dozens or even hundreds of "bands". Climate modelers consider 50 and more "variables" for each location in the atmosphere, indicating measures like temperature, humidity, wind speed, trace gases, etc.
- **null values:** is there support for null values? For single null values vs several null values? Proper treatment of null values in operations? Null values are well known in databases, and scientific data definitely require them, too. However, instrument observations typically know of more than one null value (such as "value unknown", "value out of range", "no value delivered", etc.), and these meanings typically are piggybacked on some value from the data type (such as -9999 for "unkown depth"). Such null values should be considered by array databases, too. Operations must treat null values appropriately so that they don't falsify results.
- **data integration:** can queries integrate array handling with data represented in another model, such as: Relational tables? XML stores? RDF stores? Other? This is important, eg, for data/meta-data integration - arrays never come standalone, but are ornamented with metadata critically contributing to their semantics. Such metadata typically reside already under ordered data management (much more so than the arrays themselves, traditionally) frequently utilizing some well-known data model.
- **General-purpose or domain specific?** Array databases per se are domain independent and, hence, can be used for all application domains where arrays occur. However, some systems have been crafted with a particular domain in mind, such as geo data cubes, and consequently may be less applicable to other domains, such as medical imagery.

Processing model expressiveness:

- **query language expressiveness** (built-in): This section investigates functionality which is readly available through the primary query language and directly supported by the system (i.e., not through extension mechanisms).
    - **formal semantics:** is there a mathematical semantics definition underlying data and query model? While this may seem an academic exercise a formal semantics is indispensable to verify that the slate of functionality provided is sufficiently complete (for a particular requirements set), consistent, and without gaps. Practically speaking, a well-defined semantics enables safe machine-to-machine communication, such as automatic query generation without human interference.
    - **declarative:** does the system offer a high-level, declarative query language? Low-level procedural languages (such as C, C++, Java, python, etc.) have several distinct disadvantages: (i) They force users to write down concrete algorithms rather than just describing the intended result; (ii) the server is constrained in the potential of optimising queries; (iii) delarative code can be analyzed by the server, e.g., to estimate costs and, based on this, enforce quota; (iv) a server accepting arbitrary procedural code has a substantial security hole. SQL still is the role model for declarative languages.

- **optimizable:** can queries be optimized in the server to achieve performance improvements? What techniques are available? Procedural code typically is hard to optimize on server side, except for "embarrassingly parallel" operations, i.e., operations where parallelization is straightforward. Declarative languages usually open up vistas for more complex optimizations, such as query rewriting, query splitting, etc. (See also discussion later on system architectures.)
- **subsetting** (trim, slice) operations: can arrays be subset along all dimensions in one request? Extraction of sub-arrays is the most fundamental operation on arrays. Trimming means reducing the extent by indicating new lower and upper bounds (which both lie inside the array under inspection) whereas slicing means extracting a slab at a particular position on an axis. Hence, trimming keeps the number of dimensions in the output while slicing reduces it; for example, a trim in x and y plus a slice in t would extract, from a 4-D x/y/z/t datacube, a 3-D x/y/z timeslice. Systems must support server-side trimming and slicing on any number of dimensions simultaneously to avoid transporting excessive amounts of data.
- **common operations:** can all (unary and binary) operations which are available on the cells type known to the system also be applied element-wise to arrays? Example: a+b is defined in numbers, so A+B should be possible on arrays.
- **array construction:** can new arrays be created in the databases (as opposed to creating arrays only from importing files)? For example, a histogram is a 1-D array derived from some other array(s).
- **aggregation operations:** can aggregates be derived from an array, supporting common operations like sum, average, min, max? Can an aggregation query deliver scalars, or aggregated arrays, or both? Note that aggregation does not always deliver just a single number - aggregation may well just involve selected axes, hence return a (lower-dimensional) array as a result.
- **array joins:** can two or more arrays be combined into a result array? Can they have different dimensions, extents, cell types? While such functionality is indispensable (think of overlaying two map images) it is nontrivial to implement (think of diverging partitioning array schemes), hence not supported by all systems.
- **Tomlin's Map Algebra support:** are local, focal, zonal, global operations [43] expressible in queries. Essentially, this allows to have arithmetic expressions as array indexes, such as in "a[x+1] - a[x-1]". Image filtering and convolution is maybe the most prominent application of such addressing, but there are many important operations requiring sophisticated array cell access – even matrix multiplication is not trivial in this sense.

- **external function invocation**: can external code (also called UDF, User-Defined Functions) be linked into the server at runtime so that this code can be invoked from within the query language? Commonly, array query languages are restricted in their expressiveness to remain "safe in evaluation". Operations more complex or for which code is already existing can be implemented through UDFs, that is: server-side code external to the DBMS which gets linked into the server at invocation time. Obviously, UDFs can greatly enhance DBMS functionality, e.g., for adding in

domain-specific functionality. Some systems even implement core array functionality via UDFs. To avoid confusion we list built-in and UDF-enabled functionality separately.

Import/export capabilities:

- **Data formats:** what data formats are supported, and to what degree?
- **ETL tools:** what mechanisms exist to deal with inconsistent and incomplete import data?
- **Updates to regions within arrays:** How selectively can array cells be updated? The (usually massive) arrays need to be built piecewise, and sometimes need to be updated in application-dependent areas; for example, a road map raster layer may need to be updated exactly along the course of a road that has been changed, defined maybe through some polygonal area.

Client interfaces:

- **Domain-independent interfaces:** which domain-independent interfaces exist for sending queries and presenting results?
- **Domain-specific interfaces:** which domain-specific clients exist for sending queries and presenting results?

Functionality beyond arrays: can queries perform operations involving arrays, but transcending the array paradigm? This section is a mere start and should be extended in future. However, at the current state of the art it is not yet clear which generic functionality is most relevant.

  - **polygon/raster clipping:** Can a clipping (i.e., join) be performed between raster and vector data? Such functionality is important in brain research (ex: analyze brain regions defined in some atlas), in geo services (ex: long-term vegetation development over a particular country), and many more applications. Sometimes such clipping is confined to 2-D x/y, but some engines allow n-D polygons.

Standards support: Which array service standards does the tool support? Currently, two standards are particularly relevant for arrays or “datacubes”:

- **ISO SQL 9075 Part 15: Multi-Dimensional Arrays (MDA)** extends the SQL query language with domain-neutral modeling and query support for n-D arrays [26], adopting the rasdaman query model [34]. As an additional effect, SQL/MDA establishes a seamless integration of (array) data and (relational) metadata which is seen as a game changer for science and engineering data.
- **OGC Web Coverage Processing Service (WCPS)** defines a geo datacube analytics language [13] [14]. Its core principles are similar to SQL/MDA, with two min differences. First, WPCS knows about geo semantics, understanding spatial and temporal axes, coordinate reference systems (and transformations between them). It is based on the OGC datacube standard which centers around the model of spatio-temporal *coverage* data [36]. Second, it is prepared for integration with XPath/XQuery as most metadata today are stored in XML. Experimentally, such an integration has already been performed [30]. Within the EarthServer initiative, WCPS has demonstrated its capabilities on Petabyte datacube holdings [3].

7.2.2 Feature Matrix

	Array DBMS						
	full-stack Array DBMS				Add-on array support		
	rasdaman	**SciDB**	**SciQL**	**EXTASCID**	**PostGIS Raster**	**Oracle GeoRaster**	**Teradata Arrays**
Data model							
dimensions	n-D	n-D	n-D	n-D	2D	2D	1..5-D
array extensibility	all axes, lower and upper bound	all axes, lower and upper bound	all axes, lower and upper bound	?	X & Y axes, lower and upper bound	yes	no
cell data types	int, float, complex, structs	numeric types, datetime	Any SQL data type	? (presumably C++ primitive types)	int, float, band-wise structs	int & float (various lengths), structs	common SQL data types (except var-iable length)
null values	yes, null velue sets and intervals, can be assigned dynamically	yes (single null)	yes, SQL-style (single null)	?	yes (single value)	yes, SQL-style (single value)	yes, SQL-style (single value); defined at table creation time
Data integration							
relational tables	yes, via SQL/MDA std	no	yes	yes	yes, via postgresql	yes	yes
XML stores	yes, via WCPS std	no	no (MonetDB/XQuery is not maintained since 2011)	no	yes, via postgresql	yes	yes
RDF stores	yes, with AMOS II	no	yes	no	Only via postgresql plugins	yes	yes

Other					OSM, OGR		
Domain specific?	generic	generic	generic	generic	geo raster	geo raster	generic
horizontal spatial axes	yes	no	no	no	yes	yes	no
height/depth axis	yes	no	no	no	no	no	no
time axis	yes	np	no	no	no	no	no
Processing model							
query language expressiveness (built-in)	declarative array QL	declarative array QL	declarative array QL	no, function calls	array functions with specific microsyntax, not tightly integrated with SQL	PL/SQL + object-relational functions with sub-language	array functions with specific microsyntax, not tightly integrated with SQL
formal semantics	Array Algebra	no	no	no	no	no	no
tightly integrated with SQL or some other QL	yes, via SQL/MDA std	no	yes	no	array 'Map Algebra' syntax separate from SQL	no	no
optimizable	yes	yes	yes	no	yes	no (array functionality not integrat-ed with QL)	no
subsetting (trim, slice)	yes	yes	yes	no	yes	trim	yes
common cell operations	yes	yes	yes	no	yes	yes	yes
arbitrary new array derivation	yes	yes	yes	no	yes	yes	only up to 2559 cells; initialization with literals or through UDF
aggregation	yes	yes	yes	no	yes	yes	yes

array joins	yes	yes	yes	no	yes		no
Tomlin's Map Algebra	yes	yes	on principle, via WHERE clause predicates on indexes	no	in MapAlgebra() function (only local, focal)	only local	only local
external function invocation (UDF)	yes	yes	yes	yes	yes	yes	yes
Import / export							
data formats	large number of formats: CSV, JSON, (Geo)TIFF, PNG, NetCDF, JPEG2000, GRIB2, etc.	CSV/text, binary server format	FITS, MSEED, BAM and (Geo)TIFF	?	large number of formats, includ-ing GeoTIFF	TIFF, GIF, BMP, PNG	?
data cleansing	yes, ETL tool	no	no	?	no	no	no
array cells update	any cell or region	any cell or region	any cell or region	?	down to single cell level	down to single cell level	down to single cell level
Client interfaces							
domain-independent	python, R	python, R, julia	python, R	?	psql	PL/SQL	Teradata SQL
domain-specific	many geo clients via OGC standards: OpenLayers,	?	?	?	MapServer, GeoServer, Deegree, QGIS, ...	?	no

	QGIS, NASA WorldWind, ...						
Beyond arrays							
polygon/raster clipping	yes	no	no	no	yes (2D)	no	no
Standards support							
ISO SQL MDA	yes	no	no	no	no	no	no
OGC / ISO geo datacubes (coverages)	yes	no	no	no	no	no	no
Remarks					"when creating overviews of a specific factor from a set of rasters that are aligned, it is possible for the overviews to not align."		some funct-ionality only on 1D arrays; array size limited to less than 64 kB, array generat-ion to 2559 cells; array operators in function syn-tax, no infix (like "a+b");

Array tools									
OPeNDAP Hyrax	**xarray**	**Tensor-Flow**	**Wendelin .core**	**Google Earth Engine**	**Open Data Cube**	**xtensor**	**boost:: geometry**	**Ophidia**	**TileDB**

Data model										
Dimensions	n-D	N-D	N-D	n-D	2-D	2-D, 3-D	N-D	N-D	N-D	n-D
array extensibility	no	yes	all axes, in-memory	yes	?	yes	all axes, in-memory	all axes, in-memory	yes	yes
cell data types	numeric types	docs unclear, assuming same as numpy	int, float, string, bool, structs	python numeric data types	likely various numeric types	netCDF cell data types	C++ data types	C++ data types	C prim-itives?	Num-eric types, fixed array, variab-le array, string
null values	no	yes	yes (placeho lders)	no	no	no	no	no	?	yes
Data integration										no
relational tables	yes	no	no	no	no	no	no	no	no	no
XML stores	yes	no	no	no	no	no	no	no	no	no
RDF stores	yes	no	no	no	no	no	no	no	No	Key-value store
other										
Domain specific?	generic	generic	machine learning	generic	geo raster	geo raster	astrono my	generic	generic	no

horizontal spatial axes	yes	yes	no	no	yes	yes	no	no	No	no
height/depth axis	?	yes	no	no	no	no	no	no	No	no
time axis	yes	yes	no	no	no	yes	no	no	No	
Processing model										no
query language expressiveness (built-in)			no, python library	no, python library	no, functional calls, python and JavaScript	no, client-side python calls	no, C++ library	no, C++ library	no, client-side command line or python	no
formal semantics	no	no, python	no	no	no	no	no	no	no	no
tightly integrated with SQL or some other QL	no	no	no	no	no	no	no	no	no	no
optimizable	no	no	no	no	to some extent (see physical model)	no	no	yes	yes	yes
subsetting (trim, slice)	yes	no	yes	yes	yes (function call)	yes, through client-side python	yes	In-memory	yes	no
common cell operations	no	yes	yes	yes	yes (function call)	yes, through client-	yes	In-memory	yes	no

						side python				
arbitrary new array derivation	no	yes	yes	yes	yes (function call)	yes, through client-side python	yes	In-memory	no	no
aggregation	yes, with NcML	yes	yes	yes	yes (function call)	yes, through client-side python	yes	In-memory	yes	no
array joins	no	yes	yes	no	yes (function call)	no	yes	yes, main memory	yes, INTER-CUBE operat-ion; re-quires identical tiling of both arrays	no
Tomlin's Map Algebra	no	yes	yes, through python user code	no	only local	yes, through client-side python	no	no	no	no
external function invocation (UDF)	no	yes	yes, via python user code	yes, via python user code	not invocatio n from within EE functions, but through	no	yes, via C++ user code	yes, via C++ code	yes, via shell or python	no

					own wrapping code in host language					
Import / export										
data formats	Import: csv, dap-reader, dsp, ff, fits, gdal, h5, hdf, hdf4/5, … Export: ascii, net-CDF, Bin-ary (DAP), xml	large number of formats, anything that python can understan d through a library	Export: binary check-point files (state) + Saved-Model; import from same	no	GeoTIFF	netCDF	no	import requires external code	FITS, NetCDF, JSON	no
data cleansing	yes	no	No	no	upload of massive data through Google	yes	no	no	?	yes
array cells update	no	any cell or region	any cell or region	any cell or region	down to single cell level	no update function-ality	any cell or region	down to single cell	no update function ality	yes
Client interfaces										
domain-independent	C API, Web request	Python	python, c++, java, go	python, C, Fortran	?	python API	C++	C++	python	C++

	interface									
domain-specific	OGC WCS standard	No	no	no	?	?	no	no	?	no
Beyond arrays										
polygon/raster clipping	no	?	no	no	yes, 2D	no	no	yes	no	no
Standards support										
ISO SQL MDA	no	no	no	no	no	no	no	no	no	no
OGC / ISO geo datacubes (coverages)	WCS 2.0	no	no	no	no	no	no	no	no	no

	MapReduce	
Data model		
Dimensions	N-D	N-D
array extensibility	all axes	all axes
cell data types	int	Bool, int, float, complex, structs
null values	yes	Yes
Data integration		
relational tables	no	no
XML stores	no	no
RDF stores	no	no

other	-	-
Domain specific?	generic	generic
horizontal spatial axes	yes	yes
height/depth axis	yes	yes
time axis	yes	yes
Processing model		
query language expressiveness (built-in)	yes, functional	no, transformations and actions
formal semantics	yes	no
tightly integrated with SQL or some other QL	no	no
optimizable	yes	yes
subsetting (trim, slice)	yes	yes
common cell operations	?	yes
arbitrary new array derivation	?	yes
aggregation	yes	yes
array joins	no	no
Tomlin's Map Algebra	no	no
external function invocation (UDF)	no	yes, via Java code
Import / export		

data formats	NetCDF, HDF	NetCDF, HDF, CSV
data cleansing	no	no
array cells update	?	?
Client interfaces		
domain-independent	Java	Java, python
domain-specific	no	no
Beyond arrays		
polygon/raster clipping	no	not built in
Standards support		
ISO SQL MDA	no	no
OGC / ISO geo datacubes (coverages)	no	no

7.3 Tuning and Optimization

7.3.1 Criteria

This level defines how data are managed internally, including storage management, distribution, parallel processing, etc. We have looked at both automatic mechanisms (summarized under optimization) and administrator (or even user) accessible mechanisms to influence system behavior.

- **Tuning Parameters:**
  - **Partitioning** is indispensable for handling arrays larger than server RAM, and even larger than disk partitions. Some systems perform an automatic partitioning, others allow administrators to configure partitioning, maybe even through a dedicated storage layout language [6] – which obviously is advantageous given the high impact of partitioning on query performance [23].
  - **Compression:** This includes both lossless and lossy compression techniques. Depending on the data properties, lossless compression may have little or gigantic impact. For example, natural images compress to about 80% of their original volume whereas thematic map layers (which essentially are quite sparse binary masks) can compress to about 5%. Lossy compression may be offered, but is dangerous as it may introduce artifacts – think inaccuracies – at tile boundaries.
  - **Distribution** of either complete arrays or the tiles of an array enables horizontal scaling, at the price of dynamic reassembly. In particular, join operations have to be crafted carefully to maintain satisfying performance. Therefore, service operators should be able to influence placement of arrays and their partitions.
  - **Caching:** as always in databases, caching can accomplish a significant speed-up. Distinguishing factors are: what can be cached and reused – only complete results, or also intermediate results? Does cache content have to be matched exactly, or can approximate cache hits be reused?
- **Optimization techniques:**
  - **Query rewriting:** as explained earlier, replacing query expressions by some more efficient method can have a significant impact; further, it frees users from thinking about the most efficient formulation. Note that this mechanism requires a query language with runtime analysis of incoming code.
  - **Common subexpression elimination** means that the query engine is able to spot identical parts within query and evaluate them only once, rather than every time the identical subexpression appears. Again, this frees users from thinking about the most efficient way of writing their queries.
  - **Cost-based optimization** estimates the cost of answering a query before actually executing it. There is a wide field of opportunities, with a huge potential of improving response times. For example, when performing a distributed join “a+b” where both arrays are sitting on different nodes – possibly even connected through a high-latency wide-area networks – then it can make a significant difference whether array a is transported

to array b, or b gets transported to a, or a shared approach is pursued. A decision can be made base on the actual tiling of both arrays, among other impact factors [5].

- **Just-in-time compilation** of incoming queries generates CPU code that subsequently is executed for answering the query. Obviously, such machine code is substantially faster than interpreting the query or some script code, like python. It can even be substantially faster than precompiled C++ code. This principle can be extended to generating target code for **multiple cores** and for **mixed target hardware**, such as CPU and GPU.
- Notably, all the above techniques can be combined advantageously through an intelligent optimizer.

7.3.2 Feature Matrix

	Array DBMS						
	full-stack Array DBMS				Add-on array support		
	Rasdaman	**SciDB**	**SciQL**	**EXTASCID**	**PostGIS Raster**	**Oracle GeoRaster**	**Teradata Arrays**
Tuning Parameters							
partitioning	any nD tiling	regular nD chunking	no	any nD chunking	small arrays (100x100 recommended), query to explicitly manage assembling larger arrays from tiles	yes (during raster creation)	no
compression	several lossy and lossless methods (zlib, RLE, CCITT G4, wavelets, ...)	RLE	no	no	no	yes (JPEG, DEFLATE)	no
distribution	automatic query distribution, peer federation (shared nothing)	yes (shared-nothing)	no	yes (shared-memory, shared-disk servers as well as shared-nothing clusters)	no	yes	no
caching	yes, can reuse approximate matches	yes, persistent chunk caching, temporary result caching (exact match)	?	no	no	yes	no
Optimization							
query rewriting	yes, ~150 rules	yes	yes	no	no	no	no
common subexpression	yes	?	?	no	no	no	no

elimination							
cost-based optimization	yes	?	?	no	no	no	no
just-in-time query compilation, mixed hardware	yes	no	no	no	no	no	no

	Array tools									
	OPeN-DAP	**xarray**	**Tensor-Flow**	**wendelin.core**	**Google Earth Engine**	**OpenData Cube**	**xtensor**	**boost::geometry**	**Ophidia**	**TileDB**
Tuning Parameters										
partitioning	yes, as per NetCDF	no	no	maybe indirectly, via NEO ZODB	no	no		no	no	regular tiling
compression	yes, as per NetCDF	no	sparse tensor	no	no	no		no	yes zlib)	yes, per tile

distribution	No	no	yes, with Cloud ML	maybe indirectly, via NEO ZODB	no	no		no	yes	yes, if the under-lying VFS supports it like HDFS does
caching	No	no	yes	yes	yes	yes		no	?	yes
Optimization										
query rewriting	No	no	no	no	no	no		no	no	no
common subexpression elimination	No	no	no	no	yes	no		no	no	no
cost-based optimization	No	no	no	no	no	no		no	no	no
just-in-time query comp., mixed hardware	No	no	no	no	no	no	no	no	no	no

	MapReduce	
	SciHadoop	**SciSpark**
Tuning Parameters		
Partitioning	yes	Yes
Compression	no	No
Distribution	yes	Yes
Caching	no	Yes

Optimization		
query rewriting	no	no
common subexpression elimination	no	Yes, implicit through caching
cost-based optimization	no	no
just-in-time query compilation, mixed hardware	no	no

7.4 Architectural Comparison

7.4.1 Criteria

This section aims at shedding some light on the high-level architecture of the systems and tools. As such, there is usually not a “better” or “worse” as in a comparative benchmark – rather, this section is of informative nature. An exception is the list of potential limitations.

- **implementation paradigm:** what is the overall architecture approach?
- **storage organization:**
  - does the system support partitioning (tiling, chunking) of arrays?
  - does the system support non-regular tiling schemes? Which ones?
  - What mechanisms does the system support for managing data partitioning?
  - can tiles of an array reside on separate computers, while the system maintains a logically integrated view on the array?
  - can the system process data maintained externally, not controlled by the DBMS?
  - Can the system process data stored in tape archives?
- **Processing & parallelism:**
  - which parallelization mechanisms does the system support: local single thread vs multicore-local vs multinode-cluster/cloud vs federation
  - does the system have a single point of failure?
  - federations
  - heterogeneous hardware support
- **Limitations:** Are there any particular known limitations?

7.4.2 Feature Matrix

	Array DBMS						
	full-stack Array DBMS				add-on Array support		
	rasdaman	**SciDB**	**SciQL**	**EXTASCID**	**PostGIS Raster**	**Oracle GeoRaster**	**Teradata Arrays**
Architecture paradigm	full-stack Array DBMS implementation	full-stack Array DBMS implementation	SQL + proprietary extension	extension to GLADE	SQL + object-relational types	Oracle proprietary	SQL + UDFs
Storage organization							
partitioning	any nD tiling	nD, regular	no	any nD tiling	done by user (and re-assembled through query)	2D, regular	no
non-regular tiling	any nD tiling	no	no	yes	yes (with manual re-assembly in query)	no	no
managing data partitioning	via query language	via query language	no	manually	via ingestion script	yes	no
tiles on separate computers		yes	no	yes	no	yes	no
processing on preeexisting archives (with their individual organization)	yes, any archive structure	no	no (data vaults come closest, but import on query)	no	yes (out-of-band)		no

tape archive access	yes	no	no	no	no	?	no
Processing & parallelism							
parallelization mechanisms	inter- and intra-query parallelization	inter- and intra-query parallelization	inter- and intra-query parallelization	via GLADE engine	none known	yes (Tomlin local operations)	no
single point of failure?	no	yes (orchestrator)	yes	?	yes	no	?
Federations	yes	no	no	no	no	no	no
heterogeneous hardware support					no		no
Remarks					recommended tile size 100x100		array size limited to less than 64 kB

	Array tools									
	OPeNDAP	**xarray**	**Tensor-Flow**	**wendelin.core**	**Google Earth Engine**	**Open-Data-Cube**	**xtensor**	**boost:: geometry**	**Ophidia**	**TileDB**
Architecture paradigm	Web frontend, based on DAP protocol, with format-specific processors in the back-	python library	python with XLA (Accelerated Linear Algebra)	python library for arrays larger than RAM	Google proprietary	python + xarray	extension to Mathematica	C++ library for main-memory array handling	MySQL + UDFs + MPI	C++ library, storage manager for dense & sparse multi-dimensio

	ground									nal arrays
Storage organization										
partitioning	yes, as per NetCDF	no (main memory centric)	no	yes, via NEO ZODB, but array agnostic	yes (typically, 256x256 pixels to match in-put pre-processing)	yes	no	no	no	Yes, regular tiling
non-regular tiling	yes, as per NetCDF	no	no	no	no	no	no	no	no	No
managing data partitioning	no	no	no	no	internally fixed, not under user control	via ingest-ion script	no	no	no	Yes
tiles on separate computers	no	no	no	yes, via NEO ZODB	yes	no	no	no	no	yes, via VFS (virt-ual file system) with dist-ribution similar to HDFS
processing on preexisting archives	no	no	no	no	no (data must sit in	no	no	no	no	no

(with their individual organization)					Google)					
tape archive	no	no	no	no	no	no	no	no	no	no
access										
Processing & parallelism										
parallelization mechanisms	no	yes	yes, various parallelization methods, CPU/GPU	no	yes (Google infrastructure)	no		no	yes ("embarrassingly parallel" operations, one by one)	Yes
single point of failure?	n.a.	yes	yes	no	?	n.a.	yes	n.a.	yes	No
federations	no	no	no		no	no	no	no	no	No
Heterogeneous hardware support	no	no	yes		no	no	no			No
Remarks		main memory	main memory				main memory of desktop			

	MapReduce	
	SciHadoop	**SciSpark**
Architecture paradigm	MapReduce	MapReduce

Storage organization		
partitioning	yes, regular tiling chosen by user, and based on the partitioning of the input data	yes, regular tiling chosen by user, and based on the partitioning of the input data
non-regular tiling	no	no
managing data partitioning	yes	yes
tiles on separate computers	Yes	yes
processing on preeexisting archives (with their individual organization)	Yes	yes
tape archive access	No	no
Processing & parallelism		
parallelization mechanisms	yes, MapReduce	yes, MapReduce
single point of failure?	yes, NameNode	yes, Spark master
federations	No	no
heterogeneous hardware support	No	yes, GPU (1)

7.5 References used

For the elicitation of the above feature matrices the following references have been used for the systems investigated:

- boost::geometry:
  - boost: http://www.boost.org/doc/libs/1_50_0/libs/geometry
- EXTASCID:
  - http://faculty.ucmerced.edu/frusu/Projects/GLADE/extascid.html
  - [17][18]
- Google Earth Engine:
  - Google: https://developers.google.com/earth-engine/
  - [25]
- OPeNDAP:
  - Opendap: http://docs.opendap.org/index.php/QuickStart
  - Opendap: https://opendap.github.io/documentation/UserGuideComprehensive.html#WWW_Interface
  - Opendap: https://opendap.github.io/documentation/UserGuideComprehensive.html#NetCDFTools
  - Opendap: https://www.unidata.ucar.edu/software/thredds/v4.5/tds/TDS.html
  - Opendap: https://www.opendap.org/support/faq/server/matlab-status
  - Opendap: https://opendap.github.io/hyrax_guide/Master_Hyrax_Guide.html
- OpenDataCube[2]:
  - ODC: https://ac.els-cdn.com/S0034425717301086/1-s2.0-S0034425717301086-main.pdf
  - ODC: http://datacube-core.readthedocs.io/en/latest/ops/config.html#ingestion-config
  - ODC: http://nbviewer.jupyter.org/github/opendatacube/datacube-core/blob/develop/examples/notebooks/Datacube_Summary.ipynb
  - ODC: https://www.slideshare.net/AmazonWebServices/earth-on-aws-nextgeneration-open-data-platforms
- Ophidia:
  - Ophidia: http://ophidia.cmcc.it/documentation/
- Oracle GeoRaster:
  - Oracle: https://docs.oracle.com/cd/B19306_01/appdev.102/b14254/geor_intro.htm
  - Oracle: https://docs.oracle.com/database/121/GEORS/basic-georaster-operations.htm#GEORS300
- PostGIS Raster:
  - PostgreSQL: https://postgis.net/docs/

[2] Open Data Cube is also known as Australian Data Cube, CEOS Data Cube, and some other names: “adoption of the AGDCv2 codebase by NASA's Systems Engineering Office for the Committee on Earth Observing Satellites (the CEOS-SEO)”

    - PostgreSQL: http://postgis.net/features/
    - PostgreSQL: http://postgis.net/docs/RT_ST_MapAlgebra.html
    - PostgreSQL: http://postgis.net/docs/manual-dev/using_raster_dataman.html#RT_Raster_Loader
- rasdaman:
    - rasdaman: www.rasdaman.org
    - [2][3][5][6][7][11][20][34][38][33][31][14][30][3]
- SciDB:
    - [16][19][42]
    - Paradigm4: https://paradigm4.atlassian.net/wiki/spaces/ESD/overview
    - Paradigm4: https://github.com/Paradigm4
- SciHadoop:
    - DAMASC research group.
- SciQL:
    - [47][27]
    - MonetDB: https://en.wikipedia.org/wiki/MonetDB
- SciSpark:
    - https://scispark.jpl.nasa.gov/
    - https://databricks.com/blog/2016/10/27/gpu-acceleration-in-databricks.html
- TensorFlow:
    - Tensorflow: https://www.tensorflow.org/get_started/
    - Tensorflow: https://cloud.google.com/ml-engine/docs/distributed-tensorflow-mnist-cloud-datalab
- Teradata Arrays:
    - Teradata: https://www.info.teradata.com/HTMLPubs/DB_TTU_14_00/index.html#page/SQL_Reference/B035_1145_111A/ARRAY_Functions.081.001.html
- wendelin.core:
    - Wendelin.core: https://lab.nexedi.com/nexedi/wendelin.core
    - Wendelin.core: https://www.nexedi.com/wendelin-Core.Tutorial.2016
    - Wendelin.core: https://lab.nexedi.com/nexedi/neoppod/blob/master/README.rst
- xarray:
    - http://xarray.pydata.org
    - http://xarray.pydata.org/en/stable/why-xarray.html
    - http://xarray.pydata.org/en/stable/generated/xarray.DataArray.dtype.html?highlight=dtype
    - http://xarray.pydata.org/en/stable/generated/xarray.DataArray.isnull.html
- xtensor:
    - xtensor: https://xtensor.readthedocs.io/en/latest/
    - xtensor: https://github.com/QuantStack/xtensor

7.6 Performance Comparison

7.6.1 Systems tested

The benchmark tests various functionalities, data sizings, and also the effect of parallelization. For this report, four systems have been measured: rasdaman, SciDB, PostGIS Raster, and Open Data Cube. These represent three Array DBMSs with different implementation paradigms; hence, the choice can be considered representative for the field. Open Data Cue was chosen as a representative of array tools based on scripting languages. Not present are MapReduce-type systems, due to resource constraints – this is left for future investigation.

Operations benchmarked challenge efficient multi-dimensional data access in presence of tiling as well as operations executed on data. For the purpose of this test, focus was on "local operations" as per Tomlin's Map Algebra, i.e.: the result pixel of an array depends on one corresponding pixel in each input array (often there is just one input array, in case of array joins there are two input arrays). Operations which take one input array and transform each pixel are often characterized as "embarrassingly parallel" because each pixel can be processed independently, which allows for an easy distribution across cores without the need for respecting Euclidean neighbourhood of pixels. That is the case for more complex operations, such as Tomlin's focal, zonal, and global operations; examples include convolution and practically all relevant Linear Algebra operations, such as matrix multiplication, tensor factorization, PCA, and the like. In ISO SQL/MDA, for example, a convolution operation on array `a` using 3x3 kernel `k` would make use of the pattern

```
mdarray  [ x(0:m), y(0:n) ]
elements mdaggregate +
         over        [ kx(-1:1), ky(-1:1) ]
         using       a[x+kx,y+ky] * k[kx,ky]
```

Once operations are not "embarrassingly parallel" there is a wide open field for implementation ingenuity to parallelize them efficiently. In a future version of this benchmark such operations should be tested in addition. Likewise, array joins become non-trivial once the input arrays to be combined convey a different tiling. While solutions have been proposed in literature, such as [5], testing this was not subject of this evaluation either. Finally, some commercial tools could not be evaluated; a special case is Google Earth Engine which only runs as a black box inside the enhanced Google infrastructure so that tool comparison on identical hardware is impossible.

Generally, while comparative benchmarks are among the results most looked at, they are at the same time particularly laborious to obtain. The author team has made a best effort to do as much comparison as possible – still, it remains a wide open field which certainly deserves further attention in future. Actually, it is planned to continue evaluation work beyond finalization of this report.

The benchmark code is available as part of the rasdaman source code at www.rasdaman.org.

7.6.2 Testing approach

The approach followed is based on and extends current literature on array database benchmarking, such as [49][48] [18][50] (in chronological order). A main consensus seems that several categories of perform-

ance factors can be distinguished, the most important being: storage access, array-generating operations, and aggregation operations. Following these categories we have established a series of test situations that can be translated directly into queries in case of Array Databases, and which need to be programmed via command line, python, or C++ code for the other tools. For each category several different types of queries have been devised:

- Binary operations combining two arrays, such as “a+b”. Which binary operator this is can be considered of less importance here – we randomly chose addition. The queries cover different array dimensions and array operands with both matching and mismatching tiles.
- Binary operations applying some scalar to an array, like “a+5”; again, we chose addition as the representative tested.
- Domain-modifying operations which do not change the array values as such, like shift, extend, and band combination (e.g., combining three images into a 3-band RGB).
- Subsetting operations involving slicing, trimming, and mixed on 2-D and 3-D arrays. While subsetting is also a domain modifying operation we put it in its own category due to its importance and versatility.
- Unary operations like sine calculation, type casting, and array aggregation.
- “Blocking” operations which require materializing the whole array before they can be evaluated.
- The CASE statement and concatenation are somewhat special operations that do not fit well in the other categories.

Each query class in turn has several variations differing in the size of the arrays involved (40 kB - 4 GB), number of tiles per array (1 – 10,000 tiles), the size of the output array, etc. The table below lists the queries, expressed in the syntax of ISO SQL/MDA.

Table 1: Array benchmark queries

ID	Description	Query
B1	Sum of the array’s elements	`MDSUM(c)`

B2	For each element in an array the result element is 1 if its value is 0, otherwise the result is the common logarithm of its value	`CASE`
		`  WHEN c = 0 THEN 1`
		`  ELSE LOG10(c)`
		`END`

B3	Cast all elements to unsigned 8-bit values	`MDCAST(c AS char)`
B4	Concatenate two arrays along the first axis	`MDCONCAT(c, c, 1)`
B5	Encode an array to TIFF	`MDENCODE(c,` `  "image/tiff")`

B6	Extend the spatial domain of an array to twice its width and height	`MDRESHAPE(c,`
		`  [ 0:MDAXIS HI(c,x)*2,`
		`  0:MDAXIS HI(c,y)*2`
		`  ]`
		`)`

B7	Add two 1-D arrays with mismatching tiles	`c + d`
B8	Add two 2-D arrays with matching tiles	`c + c`

B9	Add two 2-D arrays with mismatching tiles	`c + d`
B10	Add the average value of an array to all of its elements	`c + MDAVG(c)`
B11	Add a constant scalar value to all elements of an array	`c + 4`
B12	Add two 3-D arrays with mismatching tiles	`c + d`
B13	Calculate all percentiles	`MDQUANTILE(c, 100)`
B14	Join several arrays into a single multi-band array	`MDJOIN( c, MDARRAY MDEXTENT(c) ELEMENTS 3, c )`
B15	Scale-up (2x) an array	`MDSCALE( c, [ MDAXIS LO(c,x) : MDAXIS HI(c,x)*2, MDAXIS LO(c,y) : MDAXIS HI(c,y)*2 ] )`
B16	Shift the spatial domain by a given shift coordinate	`MDSHIFT(c, [500, -1000])`
B17	Calculate the sine of every element in an array	`SIN(c)`
B18	Subset the whole spatial domain	`c[*:*,*:*]`
B19	Select a single element at a particular coordinate	`c[5, MDAXIS HI(c,y) - 5]`
B20	Slice the first axis at a particular point	`c[5, MDAXIS LO(c,y) + 3 : MDAXIS HI(c,y) - 3 ]`
B21	Trim down both axes	`c[ MDAXIS LO(c,x) + 3 : MDAXIS HI(c,x) - 3, MDAXIS LO(c,y) + 3 : MDAXIS HI(c,y) - 3 ]`
B22	Slice the first axis of a 3-D array at a particular point	`c[ MDAXIS HI(c,z), MDAXIS LO(c,x) + 3 : MDAXIS HI(c,x) - 3, MDAXIS LO(c,y) + 3 : MDAXIS HI(c,y) - 3 ]`

7.5.3 The Benchmarks

The benchmark was run on the following systems:

- Open Data Cube 1.5.4

- PostGIS Raster 2.4.1 (all GDAL drivers enabled) on top of PostgreSQL 9.6.6
- rasdaman v9.5
- SciDB 16.9

All the Bx tests of the previous section have been executed on each system, as far as supported. Values missing indicate this – for example, test B5 performs data format encoding not available in SciDB.

Every run was repeated 10x and then averaged.

The machine on which the benchmark has been evaluated has the following characteristics:

- OS: Ubuntu 14.04
- CPU: Intel Xeon E5-2609v3 @ 1.90GHz; 2x 6-core CPUs, 16MB L3 cache, 256kB L2, 32kB L1
- RAM: 64GB DDR4 2133MHz
- Disk: SSD, read speed 520 MB/sec

7.6.4 Assessment

Results are shown in Fig. 8. Surprisingly, runtime results were quite divergent, therefore the time scale is logarithmic.

As it turns out the technology landscape around Array Databases is quite varied, ranging from full-stack from-scratch implementations over object-relational DBMS add-ons to MapReduce add-ons, and all in between. In this line-up of 19 array tools many are natively designed as a service while some of them comprise command line tools or libraries which are not complete services, but may aid in developing services. Technologies were evaluated through

- a feature walk-through addressing functionality (logical model), tuning and optimization (physical level), and architecture;
- a comparative benchmark between selected systems.

Investigation, for resource reasons, could only cover storage access and “embarrassingly parallel” operations; what is left for future research are operations whose parallelization is more involved, including general Linear Algebra and joins. Nevertheless, some interesting facts can be observed.

Overall, a clear ranking is visible with rasdaman being fastest, followed by Open Data Cube (up to 74x slower), PostGIS Raster (up to 82x slower), and SciDB (up to 304x slower), in sequence.

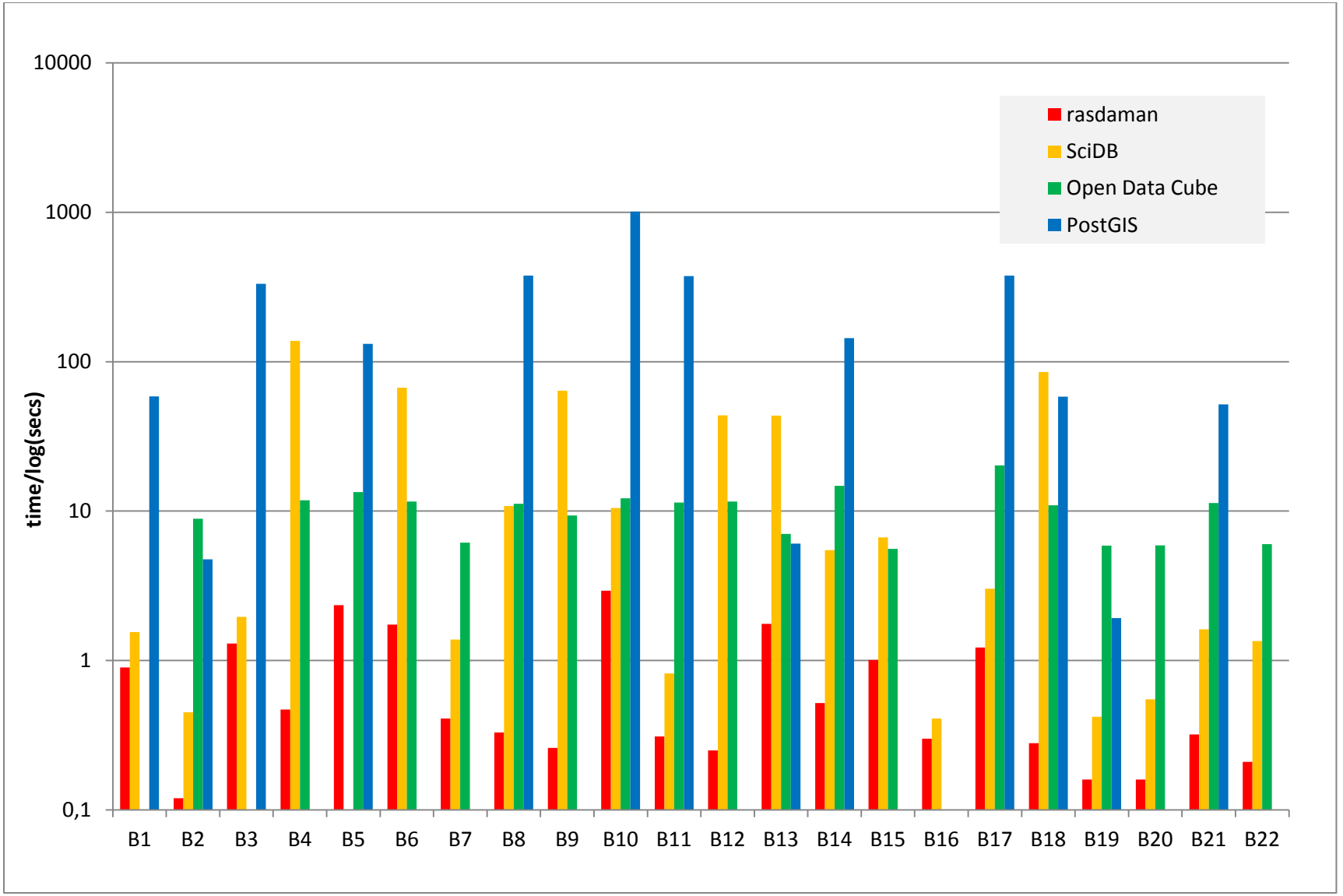

Fig. 8. Performance comparison of rasdaman, PostGIS Raster, Open Data Cube, and SciDB
(time axis logarithmic, secs; missing values represent tests not supported by the target system).

Systems offering a query language were easier to benchmark – tests could be formulated, without any extra programming, in a few lines sent to the server. Without query languages, extra programming effort was necessary which sometimes turned out quite involved. Functionality offered consisted of pre-cooked functions which may or may not meet user requirements – in this case: our test queries. Effectively, this extra burden was one reason why several systems could not be evaluated. For a system choice this means: such tools will offer only focused functionality and still leave significant burden to the user. Hence, extrapolating the notion of "analysis-ready data" we demand "analysis-ready services" which stand out through their flexibility to ask any (simple or complex) query, any time.

Compiled languages like C++ still seem to offer significant performance advantages over scripting languages like python. In a direct comparison, a C/C++ implementation was found to be faster by an order of magnitude over python code [32]. The first system, rasterio, uses python only as its frontend with C/C++ based GDAL as its workhorse. The second one, ArcPy, relies on a pure python implementation underneath, namely numpy.

UDFs can be very efficient in main memory, but general orchestration tasks of the DBMS – like storage access in face of tiling and distribution as well as allowing arbitrary queries, rather than a predefined set of UDF functionality – still remains an issue. Implementers obviously tend to prefer add-on architectures where array functionality is built on top of existing systems which offer targeted features like parallelism (such as Hadoop and Spark) or persistent storage management (like relational DBMSs). However, as these base layers are not array-aware such architectures at least today do not achieve a performance and flexibility comparable to full-stack implementations as the comparison shows.

While a hands-on evaluation of MapReduce type systems was not possible within this study there is relevant work at XLDB 2018 on a comparison of ArrayUDF (an array processing framework built on UDFs in databases, from the same group doing EXTASCID) with Spark [46]. Authors report that "In a series of performance tests on large scientific data sets, we have observed that ArrayUDF outperforms Apache Spark by as much as 2070X on the same high-performance computing system". We need to bear in mind, though, that a pure UDF without a query language constitutes just a fixed block of code performing one task – this is relatively easy to keep under control and parallelize whereas orchestration of some arbitrary query can change the performance picture substantially.

Generally, there seems to be a performance hierarchy with full-stack, from-scratch C++ implementations being fastest, followed by mixed implementations combining UDFs (read: handcrafted implementation) with a database-style orchestration engine, followed by add-ons to Hadoop / Spark, followed by object-relational add-ons.

8 Summary

With this report, RDA hopes to provide a useful basis for choosing technology when it comes to flexible, scalable analytics on massive spatio-temporal sensor, image, simulation, and statistics data. Such arrays constitute a large part of today's Big Data, forming a basic data category next to sets, hierarchies, and general graphs. In view of the known challenges in functionality, performance, scalability, and interoperability serving these arrays in a user-friendly way is a major challenge today.

Array Databases seem promising in that they provide the advantage-proven features of a declarative query language for "shipping code to data", combined with powerful techniques for efficient server-side evaluation, with parallelization being just one out of a series of known methods for speed-up and scalability.

In this study, we have provided on introduction and overview of the state of the art in Array Databases as a tool to serve massive spatio-temporal "datacubes" in an analysis-ready manner. Relevant datacube standards were listed, together with secondary information for further studies and immersion. Uptake of this report's research consists of several Big Data services with Petabyte offerings, with further ones emerging continuously. Actually, already in its preparation phase this report has found high interest; the report's Wiki access statistics indicate more than 12,000 page reads as of February 23, 2018.

The line-up of 19 different tools is an unprecedented technology overview for this emerging field. Array Databases, command line tools and libraries, as well as MapReduce-based tools have been assessed comparatively, with a clear provenance for all facts elicited. For some tools, a comparative performance analysis has been conducted showing that full-stack, clean-slate array C++ implementations convey highest performance; python constitutes a basis that comes with a performance penalty upfront, and likewise add-on implementations that reuse not array aware architectures (such as object-relational extensions and MapReduce) to emulate array support – although, admittedly, these are faster and easier to implement. Generally, implementation of the full stack of Array Databases in some fast, compiling language (like C++) pays off, although it requires a significant implementation effort.

In summary, Array Databases herald a new age in datacube services and spatio-temporal analysis. With their genuine array support they are superior to other approaches in functionality, performance, and scalability, and supported by powerful "datacube" standards. Query functionality is independent from the data encoding, and data can be delivered in the format requested by the user. Our benchmark results are in line with the increasing number of Array Database deployments on Earth science data in particular, meantime far beyond the Petabyte frontier.

With the advent of the ISO SQL/MDA standard as the universal datacube query language a game change can be expected: implementers have clear guidance, which will lead to increased interoperability (which today effectively does not exist between the systems – only one currently supports relevant standards). Applications become easily manageable across all domains, and a natural integration with metadata is provided through the SQL embedding. Further, standardization will form an additional stimulus for both open-source and proprietary tool developers to jump on this trending technology.

Such data integration will be of paramount importance in future. Standalone array stores form just another silo, even with query capabilities. It will be indispensible to integrate array handling into the metadata paradigms applications like to use. As of today, work on array integration has been done on

- **sets:** the ISO SQL/MDA standard, which is based on the rasdaman query language, integrates multi-dimensional arrays into SQL [34];
- **hierarchies:** the xWCPS language extends the OGC WCPS geo array language with metadata retrieval [29];
- **(knowledge) graphs:** first research has been done on integration arrays into RDF/SPARQL databases [2].

Still, despite its breadth, this report uncovers the need for further research. In particular, a deep comparison of the fundamentally different architectures of Array Databases and MapReduce oriented systems should be of high interest.

Obviously, Michael Stonebraker's observation of "no one size fits all" is very true also for array support – as arrays form a separate fundamental data structure next to sets, hierarchies, and graphs, they require carefully crafted implementations to deliver the usability in terms of flexibility, scalability, performance, and standards conformance which is essential for abroad uptake. Genuine Array Database technology, therefore, appears most promising for spatio-temporal datacubes, as this study indicates.

References

[1] D. Abadi: On Big Data, Analytics and Hadoop. ODBMS Industry Watch, December 05, 2012, http://www.odbms.org/blog/2012/12/on-big-data-analytics-and-hadoop-interview-with-daniel-abadi/, seen on 2018-02-25

[2] A. Andrejev, P. Baumann, D. Misev, T. Risch: Spatio-Temporal Gridded Data Processing on the Semantic Web. 2015 IEEE Intl. Conf. on Data Science and Data Intensive Systems (DSDIS 2015), Sydney, Australia, December 11-13, 2015

[3] P. Baumann, A.P. Rossi, B. Bell, O. Clements, B. Evans, H. Hoenig, P. Hogan, G.1 Kakaletris, P. Koltsida, S. Mantovani, R. Marco Figuera, V. Merticariu, D. Misev, B. Pham Huu, S. Siemen, J. Wagemann: Fostering Cross-Disciplinary Earth Science Through Datacube Analytics. In. P.P. Mathieu, C. Aubrecht (eds.): Earth Observation Open Science and Innovation - Changing the World One Pixel at a Time, International Space Science Institute (ISSI), 2017, pp. 91 - 119

[4] P. Baumann, B. Howe, K. Orsborn, S. Stefanova: Proceedings of the 2011 EDBT/ICDT Workshop on Array Databases. Uppsala, Sweden, March 25, 2011

[5] P. Baumann, V. Merticariu: On the Efficient Evaluation of Array Joins. Proc. Workshop Big Data in the Geo Sciences (co-located with IEEE Big Data), Santa Clara, US, October 29, 2015

[6] P. Baumann, S. Feyzabadi, C. Jucovschi: Putting Pixels in Place: A Storage Layout Language for Scientific Data. Proc. IEEE ICDM Workshop on Spatial and Spatiotemporal Data Mining (SSTDM), December 14, 2010, Sydney, Australia, pp. 194 – 201

[7] P. Baumann: Management of multidimensional discrete data, VLDB J., 3(4)1994, pp. 401–444, 1994

[8] P. Baumann: The Datacube Manifesto. Available on http://earthserver.eu/tech/datacube-manifesto, seen on 2018-02-25

[9] P. Baumann: Array Databases. In: T. Özsu, L. Liu (eds.): Encyclopedia of Database Systems, Springer, 2017

[10] P. Baumann: Language Support for Raster Image Manipulation in Databases. Proc. Int. Workshop on Graphics Modeling, Visualization in Science & Technology, Darmstadt/Germany, April 13 - 14, 1992

[11] P. Baumann: A Database Array Algebra for Spatio-Temporal Data and Beyond. Proc. Intl. Workshop on Next Generation Information Technologies and Systems (NGITS), July 5-7, 1999, Zikhron Yaakov, Israel, Springer LNCS 1649

[12] P. Baumann: On the Management of Multidimensional Discrete Data. VLDB Journal 4(3)1994, Special Issue on Spatial Database Systems, pp. 401 - 444

[13] P. Baumann: OGC Web Coverage Processing Service (WCPS) Language Interface Standard, version 1.0. OGC document 08-068r2, 2010

[14] P. Baumann: The OGC Web Coverage Processing Service (WCPS) Standard. Geoinformatica, 14(4)2010, pp. 447 – 479

[15] M. Blaschka, C. Sapia, G. Höfling, B. Dinter: Finding Your Way through Multidimensional Data Models. DEXA Workshop Data Warehouse Design and OLAP Technology (DWDOT'98), Vienna, Austria, August 24-28, 1998, pp. 198-203

[16]P.G. Brown: Overview of SciDB: large scale array storage, processing and analysis, in Proc. ACM SIGMOD, 2010, pp. 963–968

[17]Y. Cheng, Florin Rusu: Astronomical Data Processing in EXTASCID. Proc. 25th Intl. Conf. on Scientific and Statistical Database Management (SSDBM), 2013, pp. 47:1–47:4

[18]Y. Cheng, F. Rusu: Formal Representation of the SS-DB Benchmark and Experimental Evaluation in EXTASCID. Distributed and Parallel Databases, 2013, pp. 1–41

[19]P. Cudre-Mauroux et al: A demonstration of SciDB: a science-oriented DBMS. Proc. VLDB, 2(2)2009, pp. 1534–1537, 2009

[20]A. Dehmel: A Compression Engine for Multidimensional Array Database Systems. PhD thesis, TU München, 2002

[21]A. Dumitru, V. Merticariu, P. Baumann: Exploring Cloud Opportunities from an Array Database Perspective. Proc ACM SIGMOD Workshop on Data Analytics in the Cloud (DanaC'2014), June 2014, Snowbird, USA

[22]Gisuser: EarthServer: 1+ Petabyte Analysis-Ready Datacubes. Gis User, December 2017, http://gisuser.com/2017/12/earthserver-1-petabyte-analysis-ready-datacubes/, seen on 2018-02-25

[23]P. Furtado, P. Baumann: Storage of Multidimensional Arrays based on Arbitrary Tiling. Proc. ICDE'99, March 23-26, 1999, Sydney, Australia

[24]W. Gibson: Data, data everywhere. The Economist Special report: Managing information, 2010. http://www.economist.com/node/15557443, seen on 2018-02-25

[25]N. Gorelick, M. Hancher, M. Dixon, S. Ilyushchenko, D. Thau, R. Moore: Google Earth Engine: Planetary-scale geospatial analysis for everyone. Remote Sensing of Environment, Volume 202, 2017, pp. 18-27

[26]ISO: Information technology — Database languages — SQL — Part 15: Multi-Dimensional Arrays (SQL/MDA). ISO IS 9075-15:2017

[27]M. Ivanova, M.L. Kersten, S. Manegold: Data Vaults: a Symbiosis between Database Technology and Scientific File Repositories. Proc. SSDBM, 2012, pp. 485–494

[28]M. Koubarakis, M. Datcu, C. Kontoes, U. Di Giammatteo, S. Manegold, E. Klien: TELEIOS: a database-powered virtual earth observatory, Proc. VLDB, vol. 5, 2012, pp. 2010–2013

[29]P. Liakos, P. Koltsida, G. Kakaletris, P. Baumann: xWCPS: Bridging the Gap Between Array and Semi-structured Data. Proc. Knowledge Engineering and Knowledge Management, Springer 2015

[30]P. Liakos, P. Koltsida, P. Baumann, Y. Ioannidis, A. Delis: A Distributed Infrastructure for Earth-Science Big Data Retrieval. Intl. Journal of Cooperative Information Systems, 24(2)2015

[31]V. Liaukevich, D. Misev, P. Baumann, V Merticariu: Location and Processing Aware Datacube Caching. Proc. 29th Intl. Conf. on Scientific and Statistical Database Management (SSDBM '17). ACM, New York, USA, Article 34

[32]M. Marek-Spartz: Comparing Map Algebra Implementations for Python: Rasterio and ArcPy. Volume 18, Papers in Resource Analysis. 14pp. Saint Mary's University of Minnesota Central Services Press, http://www.gis.smumn.edu/GradProjects/Marek-SpartzM.pdf, seen on 2018-02-25

[33]G. Merticariu, D. Misev, P. Baumann: Measuring Storage Access Performance in Array Databases. Proc. 7th Workshop on Big Data Benchmarking (WBDB), December 14-15, 2015, New Delhi, India

[34]D. Misev, P. Baumann: Enhancing Science Support in SQL. Proc. Workshop on Data and Computational Science Technologies for Earth Science Research (co-located with IEEE BigData), Santa Clara, US, October 29, 2015

[35]J.H.P. Oosthoek, A.P. Rossi, P. Baumann, D. Misev, P. Campalani: PlanetServer: Towards online analysis of planetary data. Planetary Data, 2012.

[36] P. Baumann, E. Hirschorn, J. Maso, V. Merticariu, D. Misev: All in One: Encoding Spatio-Temporal Big Data in XML, JSON, and RDF without Information Loss. Proc. IEEE International Workshop on Big Spatial Data (BSD 2017), Boston, 11 December 2017

[37]PostGIS: PostGIS Raster manual. http://postgis.net/docs/manual-dev/using_raster_dataman.html, seen on 2018-02-25

[38]B. Reiner, K. Hahn: Hierarchical Storage Support and Management for Large-Scale Multi-dimensional Array Database Management Systems. Proc. DEXA, Aix en Provence, France, 2002

[39]G. Ritter, J. Wilson, J. Davidson: Image algebra: An Overview. Computer Vision, Graphics, and Image Processing, 49(1):297-331, 1990

[40]S. Sarawagi, M. Stonebraker: Efficient Organization of Large Multidimensional Arrays. Proc. Intl. Conf. on Data Engineering ICDE, Houston, USA, 1994, pp. 328-336

[41]E. Soroush, M. Balazinska, D. Wang: ArrayStore: A Storage Manager for Complex Parallel Array Processing. Proc. ACM SIGMOD, Athens, Greece, 2011, pp. 253 – 264

[42]M. Stonebraker, P. Brown, D. Zhang and J. Becla: SciDB: A Database Management System for Applications with Complex Analytics. Computing in Science & Engineering, vol. 15, no. 3, May-June 2013, pp. 54-62

[43]D. Tomlin: A Map Algebra. Harvard Graduate School of Design, 1990.

[44]Teradata: User‑Defined Data Type, ARRAY Data Type, and VARRAY Data Type Limits. https://www.info.teradata.com/HTMLPubs/DB_TTU_14_00/index.html#page/SQL_Reference/B035_1141_111A/appc.109.11.html, seen on 2018-02-25

[45]P. Webster: Supercomputing the Climate: NASA's Big Data Mission. CSC World Computer Sciences Corporation, 2012

[46]J. Wu: ArrayUDF Explores Structural Locality for Faster Scientific Analyses. XLDB, Stanford, USA, April/May 2018, https://conf.slac.stanford.edu/xldb2018/event-information/lightning-talks#mon4, seen on 2018-02-25

[47]Y. Zhang, M.L. Kersten, M. Ivanova, N. Nes: SciQL, Bridging the Gap between Science and Relational DBMS. Proc. IDEAS, 2011, pp. 124 – 133

[48]P. Baumann, H. Stamerjohanns: Towards a Systematic Benchmark for Array Database Systems. Workshop on Big Data Benchmarking (WBDB'2012), December 2012, Pune, India, Springer LNCS 8163

[49]S. Stancu-Mara, P. Baumann:A Comparative Benchmark of Large Objects in Relational Databases. Proc. IDEAS 2008, Coimbra, Portugal, November 2008

[50]G. Merticariu, D. Misev, P. Baumann: Towards a General Array Database Benchmark: Measuring Storage Access Performance in Array Databases. Proc. 7th Workshop on Big Data Benchmarking (WBDB), December 2015, New Delhi, India