Baumann *et al. J Big Data* (2021) 8:28
https://doi.org/10.1186/s40537-020-00399-2

Journal of Big Data

SURVEY PAPER Open Access

Array databases: concepts, standards, implementations

Peter Baumann, Dimitar Misev, Vlad Merticariu and Bang Pham Huu*

*Correspondence:
b.phamhuu@jacobs-university.de
Large-Scale Scientific Information Systems Research Group, Jacobs University, Bremen, Germany

Abstract

Multi-dimensional arrays (also known as raster data or gridded data) play a key role in many, if not all science and engineering domains where they typically represent spatio-temporal sensor, image, simulation output, or statistics "datacubes". 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 not keep up with the increasing requirements on performance and 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 and 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. Several papers have compared models and formalisms, and benchmarks have been undertaken as well, typically comparing two systems against each other. While each of these represent valuable research to the best of our knowledge there is no comprehensive survey combining model, query language, architecture, and practical usability, and performance aspects. The size of this comparison differentiates our study as well with 19 systems compared, four benchmarked to an extent and depth clearly exceeding previous papers in the field; for example, subsetting tests were designed in a way that systems cannot be tuned to specifically these queries. It is hoped that this gives a representative overview to all who want to immerse into the field as well as a clear guidance to those who need to choose the best suited datacube tool for their application. This article presents results of the Research Data Alliance (RDA) Array Database Assessment Working Group (ADA:WG), a subgroup of the Big Data Interest Group. It has elicited the state of the art in Array Databases, technically supported by IEEE GRSS and CODATA Germany, to answer the question: how can data scientists and engineers benefit from Array Database technology? As it turns out, Array Databases can offer significant advantages in terms of flexibility, functionality, extensibility, as well as performance and scalability—in total, the database approach of offering "datacubes" analysis-ready heralds a new level of service quality. Investigation shows that there is a lively ecosystem of technology with increasing uptake, and proven array analytics standards are in place. Consequently, such approaches have to be considered a serious option for datacube services in science, engineering and beyond. Tools, though, vary greatly in functionality and performance as it turns out.

Key words: Arrays, Array databases, Datacubes, SQL/MDA, OGC WCPS

Introduction

As The Fourth Paradigm puts it [51], "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". This reflects the insight that a meaningful structuring of data, together with suitable access methods, is instrumental for any data analysis done in any domain, including business, science, and engineering.

Since the advent of databases with high-level, declarative access interfaces [34], tabular data organization has prevailed, supported by the relational data model and query standard, SQL [57]. Long this was considered adequate for managing employees in enterprises and metadata about measurements in science, until further use cases—such as Computer Aided Design and Manufacturing (CAD/CAM) data management [28] and Computer Graphics [52] with their need for hierarchies—provoked thoughts about how to support these through data structures and query operators, too. In response, hierarchical data models were proposed. In a similar way, new requirements also triggered the need for general graph support in large-scale databases; main drivers here have been ontologies requiring comparatively small, heterogeneous graphs [49] and social networks with their large, homogeneous graphs [105]. A further relevant data structure is comprised by multi-dimensional arrays. First conceptualized in OLAP they also appear practically everywhere in science and engineering. These four data structuring principles—sets, hierarchies, graphs, and arrays (Fig. 1)—all are fundamentally different and, hence, call for dedicated database modeling and querying support, following Michael Stonebraker's observation of " no one size fits all" [108].

In database research, arrays have been treated systematically in the context of OLAP; however, these statistical "datacubes" are very sparse, while the majority of arrays in science and engineering, such as satellite images and weather forecasts, are dense. General array support in databases, while started early [14, 18], has become a general field of study only relatively recently [4, 13, 30, 32, 33, 36, 39, 43, 58, 61, 98, 104, 107, 109, 112, 122, 126], with a view on the multitude of hitherto unsupported domains.

The significant increase in scientific data that occurred in the past decade—such as NASA's archive growth from some hundred Terabytes in 2000 [46] to 32 Petabytes of climate observation data [119], as well as ECMWF's climate archive of over 220 Petabytes [19]—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

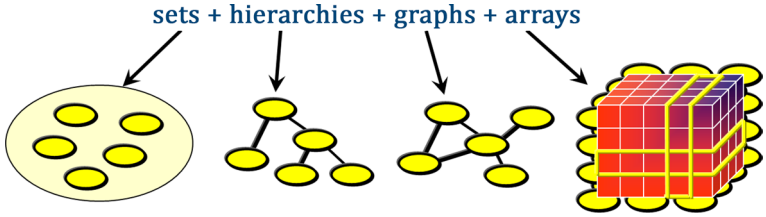

Fig. 1 Sets, hierarchies, graphs, and arrays as principal data structures in databases

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 [80] and large datacenters [19] using modern database architectures for massive spatio-temporal data sets.

Arrays—also called "raster data" or "gridded data" or, more recently, "datacubes" [21]—constitute an abstraction that appears in virtually all areas of science and engineering and 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 atmospheric and ocean data; etc.
- Life sciences: microarray data, image modalities like X-ray, sonography, PET, and fMRI deliver 2-D up to 4-D data about human and non-human brains and further organs; gene expression data come as 2-D through 4-D; etc.
- Space sciences: optical and radio telescope data; 4-D x/y/z/t cosmological simulateion output; planetary surface and subsurface data; etc.
- Statistics: "Datacubes" are known since long in the context of Data Warehousing and OLAP [25] 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).

Figure 2 gives a visual 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.

Faced with these recent developments in theory, architecture, application, and standardization in of Array Databases it is not easy to get and maintain overview. To the best of our knowledge there is no comprehensive system overview on the state of the art in this field. Benchmarks have been conducted earlier, but they typically compare only two systems [33, 35, 66, 90], as opposed to 19 in this review, and mostly without a scientifically justified benchmark design underneath (e.g., Planthaber [90]), as opposed to the benchmark design in this review which has been justified by Baumann et al. in [20].

With this technology survey this gap is to be closed. First, the approach is motivated by inspecting relevant array service standards. Also, it is motivated that availability as open-source code was not a criterion—open-source software is a business model, not a capability of the software.

As core contribution, a total of 19 relevant tools is inspected, from the fields of Array DBMSs, array command line tools and libraries, and array extensions to

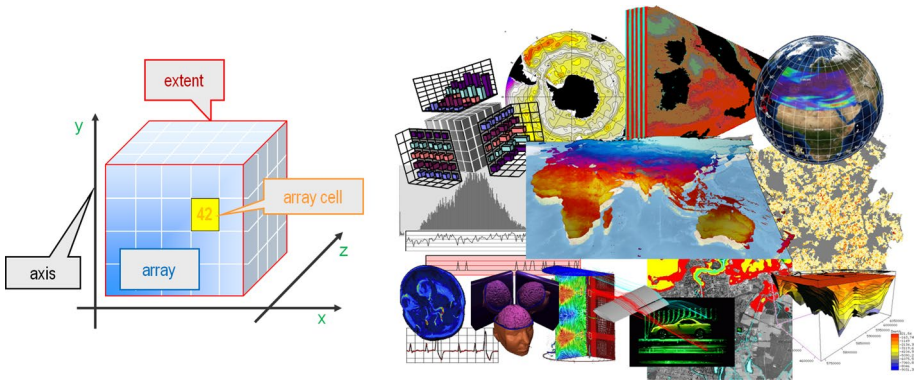

Fig. 2 Basic structure of a multi-dimensional array and its cells, with extent spanned by axes x/y/z (left); sample array data from Earth, Space, and Life Sciences as well as Engineering (right)

MapReduce-type systems. These are classified from various perspectives: functionality, standards support, and architecture.

Additionally, four Array DBMs are subject to a performance benchmark. This benchmark, which is publicly available, inspects both array data access and analysis functions applied to array data. The benchmark is designed in a way that a system cannot be tuned unilaterally; for example, cutouts walk over the array and cutouts are produced along various dimensions, always based on the same object, with one given data tiling on disk. Further, a clear distinction is made between implementation of functionality by the system itself, or implementation through User-Defined Functions (UDFs), i.e., code the service operator must provide and which obviously would not be testable. In addition to these comparisons of expressive power of the model, down-to-earth aspects have been considered, too, including array import and export capabilities, client APIs, and support for standards. Overall, more than 30 functional criteria are assessed.

Next, tuning capabilities of the systems have been inspected. Database sstems have a long tradition of providing both administrator-accessible tuning and automatic self-tuning ("optimization"). In the comparatively young field of Array DBMSs this is not always common yet and, therefore, worth investigating. Eight criteria have been inspected, considering both storage and processing optimization.

Architectural features investigated were the overall architectural paradigm, storage organization details, processing, and the parallelization approach adopted.

Necessarily, the criteria had to be adapted to the three categories of Array DBMSs, command line tools and libraries, and Hadoop-style systems, although emphasis was put on keeping criteria comparable as much as ever possible. In particular, we concentrate on array-supporting systems; for example, we investigate SciHadoop specifically, but do not look at Hadoop in general as it does not support arrays, and even less so we inspect underlying technology such as virtualization paradigms (e.g., Virtual Machines and Docker containers) nor processing models (such as CPU vs. GPU vs. quantum computing).

This investigation—consisting of agreement on the comparison criteria, collecting and analyzing 19 systems and benchmarking four systems – has been carried out produced by over a timeframe of 2 years by the survey authors in the context and with the

help of the Research Data Alliance (RDA) Array Database Assessment Working Group (ADA:WG) from which the original report is available [16]. In summary, the main contributions of this article are the following:

- A general presentation of an array model, generic enough to form a basis for a comparison of heterogeneous array systems.
- An overview on Array DBMSs and further systems offering arrays as a service, with a detailed feature comparison for 19 such systems.
- A systematic benchmark with explicit design rationales, applied to four different Array DBMSs.
- An overview on standards for array services.

The remainder of this technology review is organized as follows. In the next Section we discuss the need for database support for massive arrays, introducing the concepts of array querying. An overview on Array (database) standards is given in Sect. "Array standards", followed by an overview of technology currently available in Sect. "Array technology" and a collection of publicly accessible array (database) services in Sect. "Array systems assessment". In Sect. "Case study" we provide a technical comparison of the various technologies, including a performance benchmark. Section "Conclusion" concludes the plot.

Arrays in databases

General considerations

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 language 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 between ”data” which are conceived as large, constrained to download, with no search and ”metadata” which commonly are considered small, agile, and searchable.

Still, databases have worked out some key components of a powerful, flexible, scalable data management; these principles have proven successful over decades on sets (relational DBMSs), hierarchical data (e.g., XML [118] databases), graph data (e.g., RDF 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: 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. Notably also NoSQL approaches (initially spelt out as “No SQL”, later “Not Only SQL”), while initially denying usefulness of high-level query languages, are gradually

(re-) introducing them – prominent examples include MongoDB, Hive, Pig Latin, etc. [60].

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, rather relying on high-level interfaces abstracting away the details of storage organization.

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 [6] 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 time-series, 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 [14, 18, 38, 96] and SciDB [30, 36]; Fig. 3 gives a brief outline on the early historical development of this field. Each of the systems allows, to a larger or smaller extent, 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 formulation 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 already in 2011 [13].

An array query model for databases

Array data

Arrays being ordered homogeneous collections with a multi-dimensional addressing scheme have long been supported by programming languages, dating back to languages like APL [59]. This mostly includes primitives for accessing single array elements combined with general looping constructs. Our perspective, though, is on a different level: high-level, declarative functionality where the iteration is implicit, for reasons of user friendliness and server-side optimization opportunities, the role model always being the SQL language. Also following the tradition of query

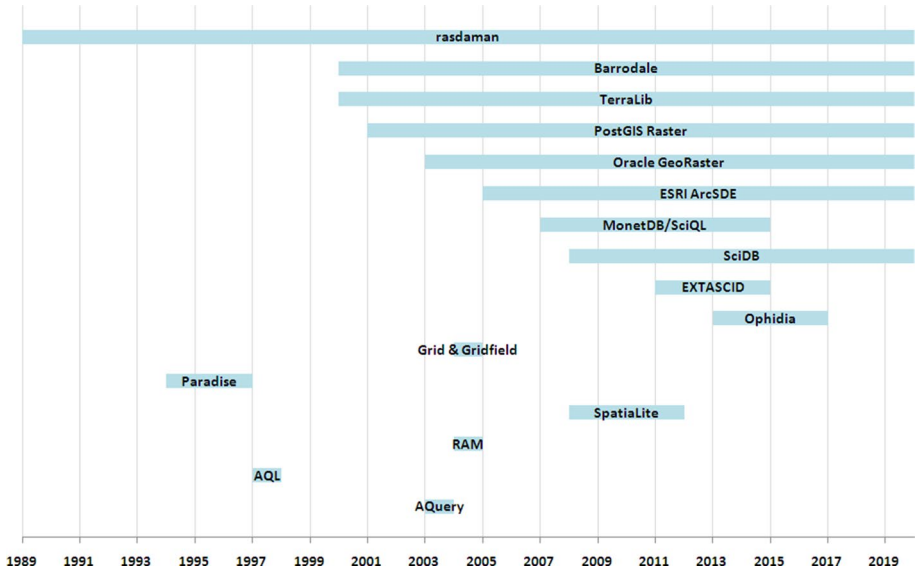

Fig. 3 Early history of array database systems

languages, an algebraic formalization of its semantics should also be available for an array query language.

Several formal models have been suggested for array databases [12]. Tomlin has established a so-called Map Algebra [117] which categorizes array operations depending on how many cells of an input array contribute to each cell of the result array. Map Algebra originally was 2-D and has been extended to 3-D meanwhile, which still is too restricted for general arrays, Further, no operational model is indicated. AFATL Image Algebra [97] has been developed to express image and signal processing as well as statistics algorithms. It is is multi-dimensional by design and has seen implementationsin libraries for various languages. Array Algebra [5] has been influenced by AFATL Image Algebra when establishing a formal framework for n-D arrays suitable for a declarative query language.

We choose Array Algebra [5] as our basis, for the following reasons. It is fully multi-dimensional; it is practically proven through implementations running on multi-Petabyte operational services [114]; it models all array operations whereas some other approaches work with black box *APPLY*() functions effectively hiding important parts of the semantics. Finally, the algebra is minimal in that only two operators allow expressing, e.g., all array operations of the SQL/MDA standard of which it is the formal basis.

We briefly present formal conceptualization of array services through Array Algebra. Readers may skip it safely, it is helpful but not strictly necessary to understand the technology analysis provided later on. Formally, a d-dimensional array is a function.

$$a : \mathrm{D} \rightarrow \mathrm{V}$$

with a domain consisting of the d-fold Cartesian cross product of closed integer intervals:

$$D = \{\mathrm{lo}_1, \ldots, hi_1\} \times \ldots \times \{lo_d, \ldots, hi_d\} \text{ with } lo_i \leq hi_i \text{ for } 1 \leq i \leq d$$

where V is some non-empty value set, called the array's *cell type*. Single elements in such an array we call *cells*. Arrays popularly are referred to as *datacubes.*

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 [117] 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 [97] is multi-dimensional by design. Array Algebra [5] has been influenced by AFATL Image Algebra when establishing a formal framework for n-D arrays suitable for a declarative query language.

Querying arrays

Although array query languages heavily overlap there is not yet a common consensus on operations and their representation. In passing we note that array operations, being 2nd order with functions as parameters, introduce functional, similar to sets, lists, and stacks. 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 inspect these in turn, based on the ISO SQL/MDA syntax.

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. 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. Figure 4 shows some examples of trimming and slicing on a 3-D array.

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 x and y:

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

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

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

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 *mdarray* operations so that the above example can be written as:

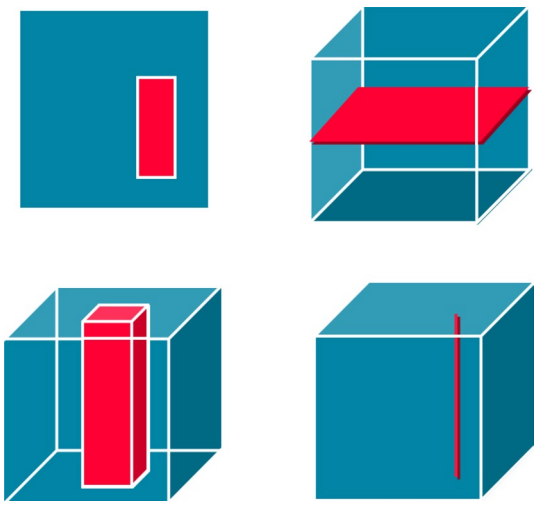

Fig. 4 Various types of subsetting from an array: trimming (left, which keeps the original dimension) and slicing (which reduces the number of dimensions, right) [41]

```
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 t_1 and t_2:

```
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 where domains A and B are "adjacent", loosely speaking,

```
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. In practice such a concatenation is used, for example, when selecting all December time slices from a climate time series.

Some systems provide an *apply*() function which receives an array expression a as first parameter and a cell-level function f, bearing the same mechanics as induced functions: apply the function to each cell of the input array. While this semantic shortcut eases specification work it has several drawbacks. On conceptual level, the semantics of f is outside the array framework, so essentially a black box. On implementation level this means the array engine has no knowledge about the behavior of the operator and, hence, cannot optimize it except for exploiting the trivial "embarrassingly parallel" property. Moreover, this is constrained to unary induced functions; while a second, binary, apply function can be supplied this does not cover the non-local operations such as histograms, convolution kernels, etc., so the larger part

of Linear Algebra. This means a significant limitation in expressiveness and, consequently, the optimization potential.

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

```
mdsum(a)
```

Not all operations can act as condensers as they must be form a monoid in order for the aggregation to work that is: the operation must be commutative and associative (this opens up parallelization opportunities) and it must have a neutral element. Common candidates fulfilling this criterion are mdsum, mdavg, mdmin, mdmax, mdexists, and mdforall.

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 a 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.

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 [67]. For example, we can define a table of Landsat satellite 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.

Array database architectures

Storage

Access patterns on arrays are strongly linked to the Euclidean neighborhood of array cells (Fig. 5), therefore it must be a main goal of any storage engine to preserve proximity

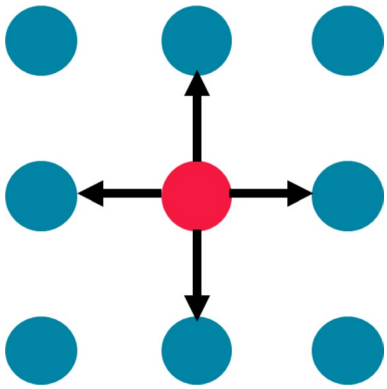

Fig. 5 Cartesian array cell neighborhood

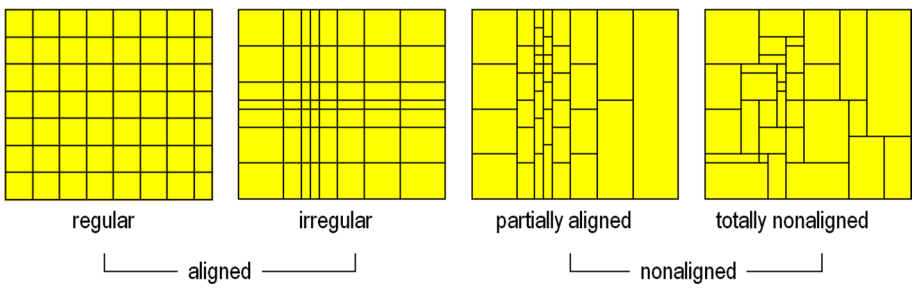

Fig. 6 Categories of array tiling patterns, after [43]

on persistent storage through some suitable spatial clustering. It is common, therefore, to partition n-D arrays into n-D sub-arrays called tiles [18] or chunks [100] which then form the unit of access to persistent storage.

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. 6). 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. Figure 6 illustrates some tiling patterns, from left to right in increasing irregularity resulting in increased adaptivity to query access patterns [43].

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 100×100 pixels [92]. Teradata arrays are limited to less than 64 kB [112]. SciDB offers a two-level partitioning where smaller partitions can be gathered in container partitions. Further, SciDB allows overlapping partitions so that queries requiring adjacent pixels (like in convolution operations) do not require reading the neighboring partitions [104]. In rasdaman, a storage layout sublanguage allows to define partitioning along several strategies [8]. 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 analysis. In the "area of interest tiling" strategy, hot spots are indicated and the system automatically determines an optimal partitioning.

To quickly determine the partitions required—a typical range query—some spatial index, such as the R-Tree [50], 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 [38]. Still, in face of very large array databases tertiary storage may be required, such as tape robots [96, 100].

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 [117] 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 [15]. Additional complexity, but also opportunities, comes with Linear Algebra operations ranging from matrix multiplication over QR decomposition up to Fourier Transform and Principal Component Analyses, 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.

Generally, parallelization in Array Databases is not constrained to the rigid "*Map*() followed by *Reduce*()" pattern of Hadoop-style systems [2, 37], but can look at each query individually and combine a wide spectrum of techniques. 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, also under the name *heuristic optimization*, looks at an incoming query to see 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. 7) 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.

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.

An extension of the principle of parallelizing query evaluation over different compute nodes is distributed processing. Additional challenges arise in that these nodes normally are independent from each other and connected through less bandwidth than in local situation. Additionally, computing and storage can be—and usually is—heterogeneous the nodes. One example of a location-transparent federation is EarthServer [114].

Still, despite this rich set of options parallelization brings along this is by no means the only opportunity for speeding up query processing. By way of example we briefly present one hardware and one software technique.

Mixed hardware Compiling queries directly into machine code for CPU, GPU, FPGA, etc. can greatly speed up processing time, even more so by dedicating tasks to the most suitable processing unit. However, mixed hardware evaluation poses non-trivial problems which still are under active research.

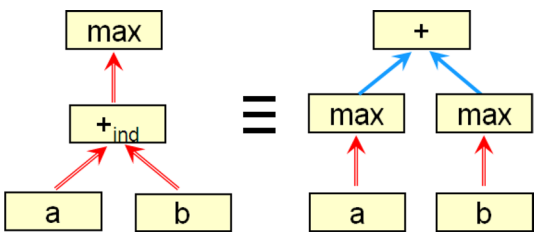

Fig. 7 Sample equivalence rule for array query rewriting: "$max(a+b) \equiv max(a)+max(b)$"

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 partially matching areas in array results [64].

Note that reformulating and compiling queries is not necessarily a time consuming task. Experience with rasdaman shows that such optimization steps altogether can be accomplished within few milliseconds.

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.

Related technology

Array databases by definition are characterized by offering a declarative query language on n-D arrays. Such technology can be 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 in itself and do not have a query concept, but rather a procedural API.

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

Array standards

Several standards relevant for large-scale array querying are already in place. Such standards may be domain independent (such as ISO Array SQL [56]) or domain specific (such as the OGC WCPS geo raster query language [22, 78]). For each standard listed its adoption status is provided.

Domain neutral standards

SQL arrays

Full title: IS 9075 SQL Part 15: Multi-Dimensional Arrays (MDA).
Issuing body: ISO/IEC SC 32/WG 3.
Description: Data and Processing Standard. SQL extension with domain-neutral definition and queries on massive multi-dimensional arrays ("datacubes").
Status: International Standard, published 2019.
Further information: [56, 67, 68].

Geo datacube standards

The main standardization body, in close collaboration with ISO, is the Open Geospatial Consortium (OGC). W3C also has done some work. While OGC and ISO operate in lock-step synchronization and mutually adopt their standards the W3C specification is different and incompatible. It should also be noted that no operational implementation of the W3C specification is known whereas the ISO/OGC standards are routinely used worldwide in large-scale deployments [114].

Geo datacube abstract concepts

Full title: Abstract Topic 6, ISO 19123-1.
Issuing body: OGC, ISO TC211.
Description: Abstract, generic data model for spatio-temporal coverages, that is: spatio-temporal regular and irregular grids, point clouds, and general meshes. This model is not intended for establishing the abstract concepts, not for implementation which is addressed by the twin specification OGC CIS/ISO 19123-2.
Status: Seasoned ISO 19123/OGC AT6 currently under rework by ISO to become 19123-1 and subsequently OGC Abstract Topic 6.
Further information: [55].

Geo datacube data model implementation

Full title: Coverage Implementation Schema (CIS), ISO 19123-2.
Issuing body: OGC, ISO TC211.
Description: Concrete, implementable, data model for spatio-temporal regular and irregular grids, point clouds, and general meshes. The model is independent from services (see below) and can be used in various OGC service types. Services using this coverage model can be conformance tested down to pixel level using the OGC compliance tests.
Status: Adopted by OGC as CIS 1.1; adopted by ISO as 19123-2 using OGC CIS 1.0 with plans to lift it to CIS 1.1 starting 2021.
Further information: [4, 9–11, 41, 77].

Geo datacube access service

Full title: Web Coverage Service (WCS).
Issuing body: OGC.
Description: WCS is a modular suite of Web service standards for accessing spatio-temporal coverages as per OGC CIS. The mandatory core specified access, subsetting, and encoding while a set of optional extensions specify additional, advanced functionality a server can provide. A detailed conformance test suite allows for free validation of implementations claiming compliance [79].
Status: Adopted OGC standard since 2012; adopted by EU INSPIRE since 2016; mid-term plans for adoption by ISO.
Further information: [7, 77].

Geo datacube analytics language

Full title: Web Coverage Processing Service (WCPS).
Issuing body: OGC.
Description: Geo datacube query language for access, analytics, and fusion over massive spatio-temporal datacubes over regular or irregular grids. WCPS is the Processing extension of the WCS suite.
Status: OGC standard since 2009; optional component in the EU INSPIRE Coverage Download Services.
Further information: [17, 41, 54, 78].

Geo datacube rDF vocabulary

Full title: RDF Data Cube Vocabulary.
Issuing body: W3C.
Description: An RDF vocabulary of datacubes for use in Semantic Web contexts with a focus on statistical data. The framework adopts aspects of the SDMX standard for statistical data exchange. Mainly metadata are modeled, single cells are not addressable, and there is no query nor processing model associated.
Status: W3C Best Practice since 2014.
Further information: [113].

Array technology

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 of 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 since epoch. 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 or insufficient knowledge exists about such standards, typically silo solutions with datacube support are established where the term "silo" illustrates insular data subsystems that are incapable of communicating and exchanging reciprocally, thereby substantially hindering insight gains [42]. 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 [103]), usually implemented in python using existing array libraries. We, therefore, also look at domain-specific "datacube" tools as well.

This review is organized 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 Big Data. After that, systems are listed which do not fall into any of the above categories. Finally, we list libraries (as opposed to the aforementioned complete engines) and n-D array data formats.

Technology overview

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 Sect. "Related technology"):

- 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* implemented from scratch (e.g., rasdaman, SciDB) and *add-ons to existing database systems* implemented as extra layers to existing DBMSs (e.g., EXTASCID), as *object-relational extensions* (ex: PostGIS Raster, Teradata Arrays, Oracle GeoRaster), or through *direct DBMS kernel coding* (e.g., 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 [37] type array engines allowing multi-dimensional array processing based on top of Hadoop [73] or Spark [74]. The underlying MapReduce paradigm relies on expressing algorithms on a map operation (such as a reordering and restructuring of input according to the criteria given) followed by a reduce operation which aggregates the restructured data. Internally, the MapReduce implementation will provide means to parallelize in the sense that the same map and reduce algorithms are executed in parallel on different cloud nodes and on different data. Algorithms which cannot be expressed this way need to resort to iterations of this map/reduce step. The map and reduce functions need to be implemented by some developer; Hadoop relies on Java, Spark uses Java and Scala. Neither the tools nor the languages used provide built-in support for arrays, except when limited to RAM of a single node. Therefore, MapReduce tools per se are out of scope of this investigation. However, array support has been added by various tools, and these we will inspect indeed.

We adopt the description of each system provided by its maintainers and augment it with own findings from inspecting the respective websites, publications, and manuals.

We first give an individual characterization as a brief overview on the systems addressed; a detailed feature-by-feature comparison can be found in Annexes 1, 2, and 3.

Generic array DBMSs

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

Rasdaman ("raster data manager") [94, 95] has pioneered Array Databases and "actionable datacubes" [5, 6, 14, 15, 17, 18, 56, 69]. It supports declarative querying of massive multi-dimensional arrays in federations of autonomous instances, 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 10 Petabytes [72], and queries have been split successfully across more than 1,000 cloud nodes [39]. The rasdaman technology is blueprint for several Big Data standards, such as ISO SQL/MDA [67] and OGC WCPS [17]. A public demonstration service is available on the Web [23].

Source code is available from [94] for the open-source *rasdaman community* edition (LGPL for client libraries, GPL for server—so can be embedded in commercial applications); the proprietary *rasdaman enterprise* edition is available from the vendor [94].

SciDB [86, 109] 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 hosting UDFs (User-Defined Functions) implementing array functionality, and also effecting parallelization.

SciDB has adopted a dual license model [87]. The source code of the community version is available from [88], although seemingly not maintained since several years; the community version is Affero: not allowed for commercial purposes.

SciQL [61, 70] was a case study extending the column-store DBMS MonetDB with array-specific operators [58, 126]. As such, n-D arrays were sequentialized internally to column-store tables (i.e., there is no dedicated storage and processing engine).

No source code could be found.

EXTASCID [32, 33, 99] is a complete and extensible research prototype for scientific data processing. It supports natively both arrays as well as relational data. Complex processing is handled by a meta-operator 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. (description taken from website below).

No source code could be found.

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 adopted 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 ("pixel", "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{ unsigned int red, green, blue;}>
```

- Further, as the SQL syntax per se 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, such as limiting to particular dimensions.

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 effort than implementing the full stack of an Array DBMS.

PostGIS raster "Raster" is a PostGIS type for storing and analyzing geo raster data [92, 93]. 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 100×100 pixels.

Source code is available under GPL v2 on the developer wiki [91].

Oracle GeoRaster [85] is a feature of Oracle Spatial allowing to 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. These data types and schema objects can be used 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, the location on Earth for a cell can be determined in an image; or given a location on Earth, the cell in an image associated with that location can be found. There is no particular raster query language underneath, nor a specific array-centric architecture.

Source code is not available as Oracle is closed source, proprietary.

Teradata arrays Teradata recently has added arrays as a datatype [111], also following an object-relational approach. There are some fundamental operations such as subsetting; however, overall the operators 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 restrictions include: Arrays must have between two and five dimensions;

only one element of the array can be updated at a time; it is unclear whether array joins are supported.

Source code is not available as Teradata is closed source, proprietary.

Array tools

OPeNDAP ("Open-source Project for a Network Data Access Protocol") is a data transport architecture and protocol for earth scientists [81]. 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.

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 provides 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.

Source code is available from [82] (license scheme not indicated).

Xarray (formerly xray) [123] is a 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. Dataset is an in-memory representation of a netCDF file.

Source code is available from [123] under an Apache license.

TensorFlow [1, 3, 110] is a tool developed by Google for machine learning. While it contains a wide range of functionality, TensorFlow is mainly designed for deep neural network models where it aims at easing creation of machine learning models for desktop, mobile, web, and cloud.

Source code is available from [110] under an Apache license.

Wendelin.core [120] allows to work with arrays bigger than RAM and local disk. Bigarrays are persisted to storage, and can be changed in transactional manner. Hence, bigarrays are similar to numpy.memmap for numpy.ndarray and operating system 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 (multi-dimensional arrays) 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 127 TB on Linux /amd64.

Source code is available from [120] under GPL v3 with specific details, see [121].

Google earth engine [47, 48] builds on the tradition of Grid systems with files, there is no datacube paradigm as such. 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. Discussion between authors and the developers revealed that Google has added a declarative Map Algebra interface in addition which resembles a subset of the rasdaman query language,

mainly induced operations and some condensers. 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.

Source code is not available, Earth Engine is closed-source, proprietary.

Open Data Cube (ODC) [75, 76] seeks to increase the value and impact of global Earth observation satellite data by providing an open and freely accessible exploitation architecture. ODC is an application layer on top of xarray and PostgreSQL, programmed in python.

Source code is available from [76] under an Apache license.

xtensor [124, 125] 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.

Source code is available from [125] under a permissive homegrown license.

Boost.Geometry (aka Generic Geometry Library, GGL), part of collection of the Boost C++Libraries, defines concepts, primitives and algorithms for solving geometry problems [26]. Boost.MultiArray provides a generic N-dimensional array concept definition and common implementations of that interface.

Source code is available from [27] under a permissive homegrown license.

The *Ophidia* framework [83][84] 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.

Source code is available from [84] under a GPL v3 license.

TileDB [115, 116] is a library managing 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.

Source code is available from [115] under a MIT license.

MapReduce-type systems

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" [2].

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

SciHadoop [31] is an experimental 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.

Source code available from [31] under a GPLv2 license.

SciSpark [101, 102] 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.

Source code is available from [102] under an Apache v2 license.

GeoTrellis [44, 45] 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.

Source code is available from [45] under an Apache v2 license.

MrGeo (pronounced "Mister Geo") is an open source geospatial toolkit designed to provide raster-based geospatial processing capabilities performed at scale [71]. MrGeo enables global geospatial big data image processing and analytics. MrGeo is built upon the Apache Spark distributed processing framework.

Source code is available from [71] under an Apache v2 license.

Array systems assessment

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 insight by Merticariu et al. [66] 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. 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 appear sufficiently similar to the ones inspected below. Examples include MrGeo and GeoTrellis as specialized Hadoop implementations offering array support.

Annex 1 lists the complete feature tables.

Functional comparison

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 at least spatio-temporal dimensions. Notably, going beyond about ten dimensions faces the curse of dimensionality, such as extreme sparsity [53].
- 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 con-

sider 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 "unknown 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, e.g., for data/metadata 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 readily 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 [117] 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 language interfaces: This addresses client-side APIs offered; while every tool will naturally support its native tool implementation language some support a range of languages, making them attractive for different purposes and different communities.

- 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 [56], adopting the rasdaman query model [67]. 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 [17, 22]. Its core principles are similar to SQL/MDA, with two main 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 [11]. 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 [63]. Within the EarthServer initiative, WCPS has demonstrated its capabilities on Petabyte datacube holdings [19].

Tuning and optimization

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 [8]—which obviously is advantageous given the high impact of partitioning on query performance [43].
- 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 improveing 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 [15].
- 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.

Annex 2 lists the complete feature tables.

Architectural comparison

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?
- Support for location-transparent federations?
- Heterogeneous hardware support?

Limitations: Are there any particular known limitations?
Annex 3 lists the complete feature tables.

Reference used

For the elicitation of the above feature matrices the references listed in this article have been used, as well as the additional sources listed in the RDA report underlying this article [16].

Performance comparison

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 engines with different implementation paradigms (with ODC being a non-database system); hence, the choice can be considered representative for the field. Open Data Cube 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 correspondingpixel 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 neighborhood 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 3×3 kernel k would

```
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]
```

make use of the pattern.

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 [15], 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 [94].

Testing approach

The approach followed is based on and extends current literature on array database benchmarking, such as [20, 33, 66, 106] (in chronological order). A main consensus seems that several categories of performance 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. 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 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 to 4 GB), number of tiles per array (1 to 10,000 tiles), the size of the output array, etc. Table 1 below lists the queries, expressed in the syntax of ISO SQL/MDA.

The benchmarks

The benchmark was run on the following systems:

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 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]*

- 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 @ 1.90 GHz; 2 × 6-core CPUs, 16 MB L3 cache, 256kB L2, 32kB L1.
- RAM: 64 GB DDR4 2133 MHz.
- Disk: SSD, read speed 520 MB/sec.

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

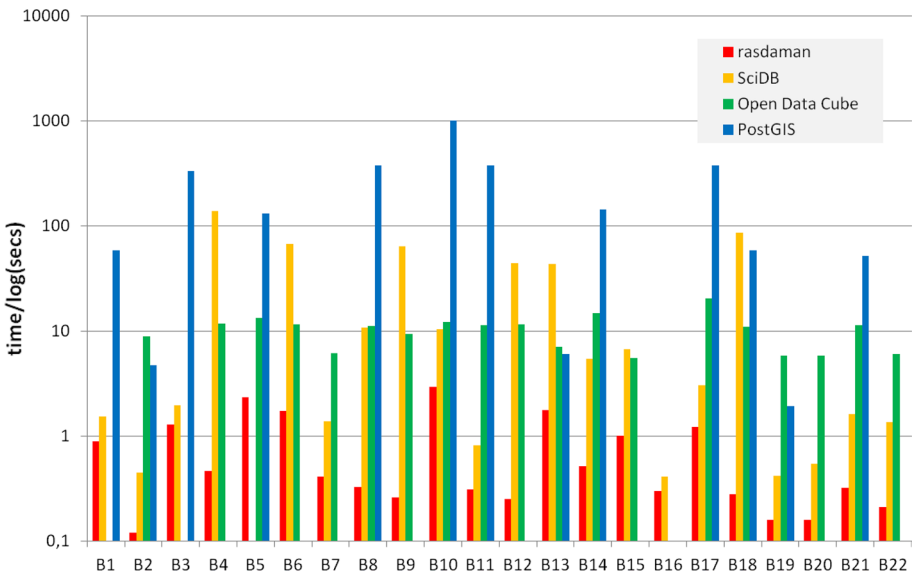

Fig. 8 Performance comparison of rasdaman, PostGIS Raster, ODC, and SciDB (time axis in seconds, logarithmic; missing values represent tests not supported by the target system)

parallelization is more involved, including general Linear Algebra and joins. Nevertheless, some interesting facts can be observed from the measurements shown in Fig. 8.

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

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 [65]. 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 when they are hand coded and optimized, but general orchestration tasks of the DBMS—like storage access in face of tiling and parallelization/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 [29] on a comparison of ArrayUDF (an array processing framework built on UDFs in databases, from the same group doing EXTASCID) with Spark [122]. 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++ systems 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.

Case study

Publicly accessible array services

In this section we provide a case study of Array DBMS services in practice. After a brief panorama of active services publicly offered we zoom in on the EarthServer datacube federation.

Overview

Below, a selection of publicly accessible services 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. 9).

A public demonstration service titled "Big Earth Datacube Standards", running rasdaman, illustrates practical use of the open standards for multi-dimensional Earth data extraction, visualization, and analysis [24]. To this end, it offers sample geo-oriented array use cases on 1-D through 5-D data sets.

PlanetServer [80, 89], a 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 powered by rasdaman community.

CODE-DE is the German Sentinel hub providing data collected by the ESA Sentinel satellite family. The batch-oriented Hadoop-based service of CODE-DE has been enhanced with interactive spatio-temporal datacube analytics using rasdaman [40].

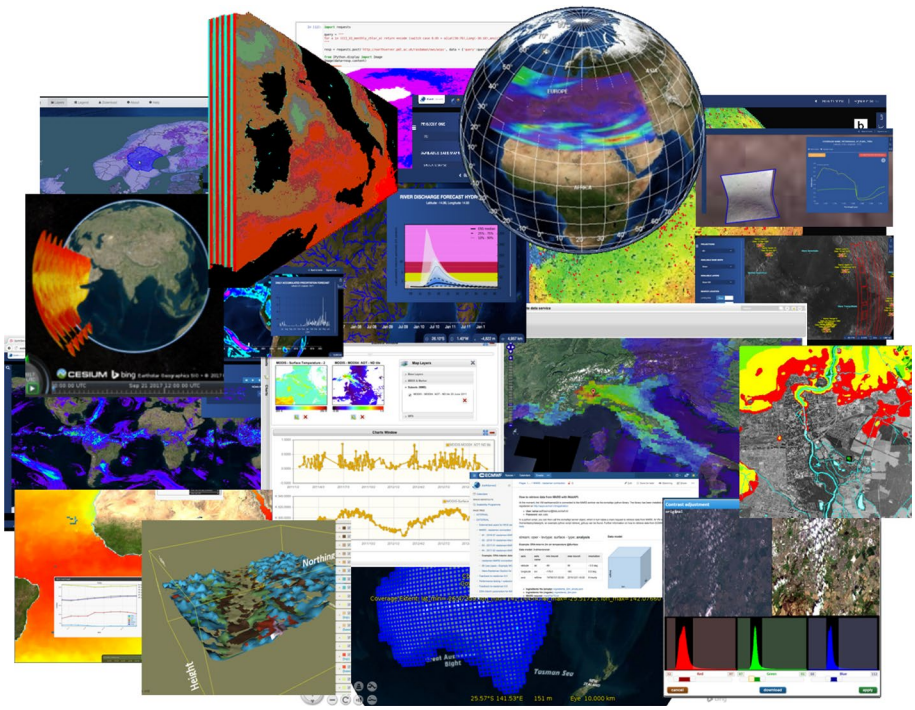

Fig. 9 Impressions of various services powered by an Array Database system (source: rasdaman/EarthServer)

EarthServer

EarthServer [114] is a federation of a growing set of large-scale Earth data providers of altogether several dozens of Petabytes. The rasdaman-backed platform offers location-transparent federation where users perceive the combined data offerings as single, homogenized information offering with free mix and match of data regardless of the individual datacube's placement.

We pick some federation members for inspection. Mundi [72] is one of the European Space Agency (ESA) satellite archives for the Sentinel fleet. This includes Sentinel-1 radar data, Sentinel-2 hyperspectral optical data at various processing levels, and Sentinel-5p providing a variety of products like aerosol index, aerosol layer height, Methane, Carbon Monoxide, Formaldehyde, and several more indicators. Each of these represents a 3-D x/y/t datacube—actually, a virtual one because the underlying files are provided in different coordinate reference systems, a fact that is hidden through a concept of virtual datacubes which act similar to relational views. Effectively, hundreds of files get virtually coalesced into one cube making handling substantially easier for users. WCPS queries allow server-side retrieval, processing, and fusion, independent from the output format chosen and the coordinate system data are stored and delivered. Every incoming query gets translated into rasql which resembles SQL/MDA modulo minor syntax differences.

The common architecture of all nodes participating in EarthServer is shown in Fig. 10. At the heart are the multi-parallel rasdaman server processes, each one individually assigned to some client. As rasdaman is domain-agnostic it does not know about space/time semantics and coordinates; this is resolved by an additional layer on top which offers Web access via the OGC API standards WMS, WCS, and WCPS.

Datacubes can be stored in BLOBs of a conventional DBMS or in rasdaman's own storage manager which is about 2× faster than, e.g., PostgreSQL as backend.

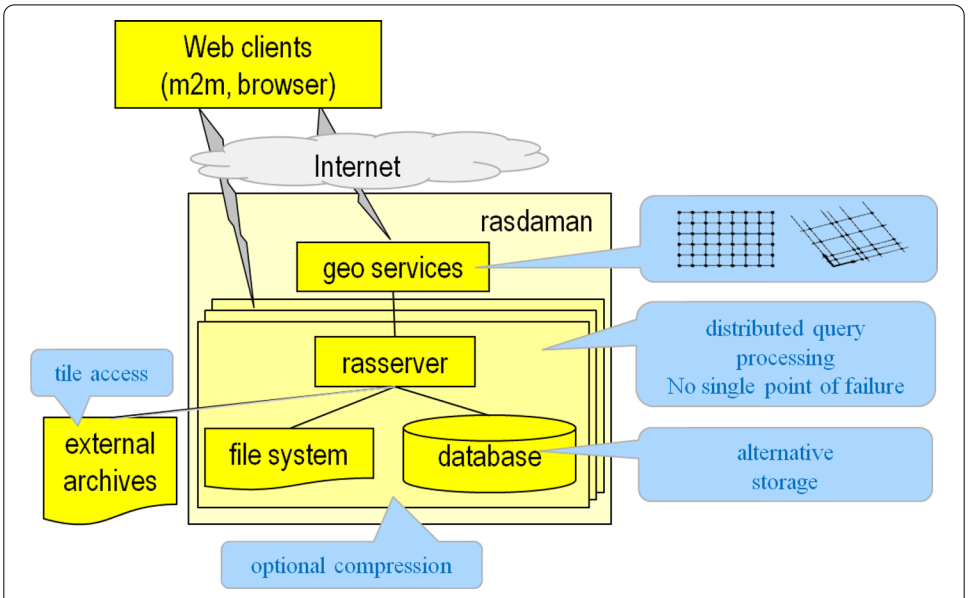

Fig. 10 EarthServer federation node high-level architecture stack

Alternatively (and used by Mundi) rasdaman can register external archives and execute queries dynamically on such data, without preloading. Obviously, in this case performance depends on how well the archive is prepared for the spatio-temporal queries submitted.

This can be studied nicely with Mundi. Satellite data files are delivered by ESA in a format called SAFE which for each image ("scene") taken consists of a zip file with metadata for coordinates and further information, plus a JPEG2000 or NetCDF file, depending on the satellite instrument type. This is exactly the format in which Mundi serves these files. From a database perspective, this is suboptimal for several reasons: Data are not under the exclusive control of the DBMS; tiling is far from optimal for timeseries analysis as every scene represents a tile of thickness 1 along time axis; due to the choice of JPEG with requires extra CPU cycles for reconstructing the original pixel. All these issues had to be addressed as import (and building up optimal structures) was not an option considering the storage costs of the Petabytes under consideration; currently, several dozens of Petabytes of datacubes are offered via Mundi.

Value of Earth data grows with variety, and so there are further datacubes representing Landsat-5, -7, and -8 as well as 2-D elevation data. All of these can be processed and combined, including remote fusion. Such queries are solved through query splitting: in a bottom-up walk of the query tree largest subtrees are built around objects all sitting on the same remote server so that these subqueries can be completely executed on that server. Optimization goals currently are (i) maximizing distributed processing and (ii) minimizing data flow between different server nodes. This principle is applied to both cluster/cloud and remote distributed setups. For the user this means complete location transparency: every query can be evaluated by every federation member, and no knowledge about an object's location is required by the user. Figure 11 illustrates this principle.

Hence, in the federation users experience a single, integrated information space abstracting away from the data center's storage and processing organization, and further particularities.

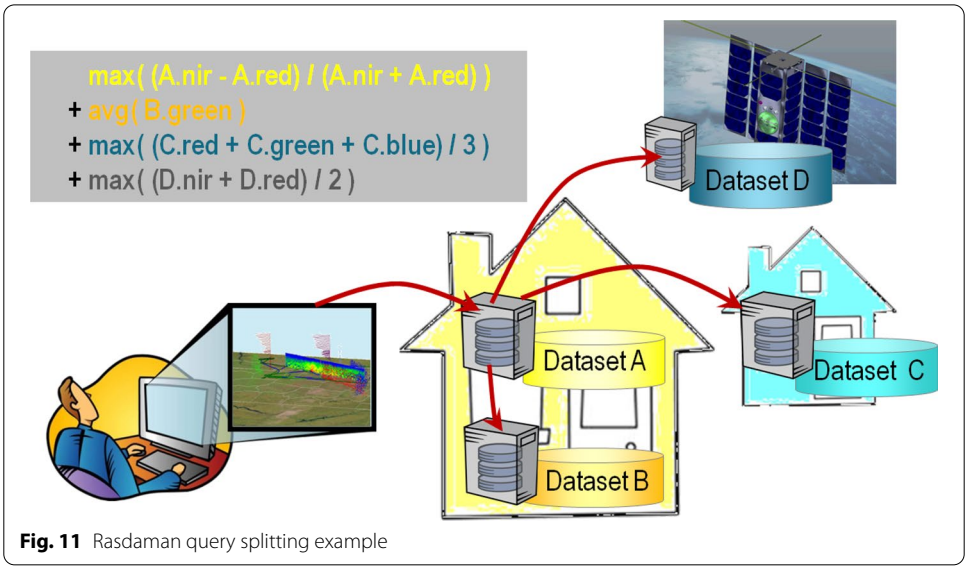

Fig. 11 Rasdaman query splitting example

It has turned out, though, that federation membership required convincing security mechanisms, securing not only access overall, but also protecting parts of datacubes; for example, the long tail of climate timeseries might be free, but the latest two weeks are available for fee only. To this end, rasdaman implements Role-Based Access Control with modifications to protect areas in datacubes down to the level of single pixels, based on bounding boxes, polygons patterns, masks, or computed criteria.

Conclusions

In this paper we have provided the following main contributions:

- A feature matrix which addresses a wide range of system properties, from abstract concepts like query language expressiveness down to practicalities such as data formats and ingestion tool support. This aids future comparative tests as a large matrix is available against which further systems can be readily compared. Array system designers get a feature list, including relevant standards, along which they can craft their own tool.
- A feature comparison of software tools offering array services, starting from the field of Array Databases and extending into neighboring fields. The expected benefit is to stimulate further research and give a decision basis for practitioners in charge of choosing the best suited datacube system.
- A public available array benchmark which is more rigorous and systematic than existing array benchmarks, and is designed to not allow tuning a system towards the tests performed, therefore enhancing its general value and reliability.
- A comparison of four Array DBMSs, which exceeds the existing test breadth where only two systems are compared. This allows for a better comparison of systems when faced with the task of choosing one.

With this survey we hope 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. Additionally, we hope it stimulates research on array service concepts and architectures, thereby advancing this comparatively young field of database research.

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.

The line-up of 19 different tools, analyzed against over 30 criteria, and 4 Array DBMSs benchmarked 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 [67];
- Hierarchies: the xWCPS language extends the OGC WCPS geo array language with metadata retrieval [62];

- (Knowledge) graphs: first research has been done on integration arrays into RDF/SPARQL databases [2]; a general graph query framework is under development in ISO.

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.

Future work should include further systems in the benchmark (in particular the MapReduce category), and also extend it with further tests for more complex queries, such as Machine Learning. It is the hope that this study has paved the way towards consolidation of functionality, but also towards a future common array service benchmark which would service providers and users enable to make more informed decisions on backend choice and service design. For system developers, the overview hopefully is helpful in determining gaps in theory and implementation, thereby contributing to further advancing the domain of array services as a whole.

Acknowledgements
The authors gratefully acknowledge the contributions made by Kwo-Sen-Kuo, NASA, as well as the helpful comments made by the reviewers.

Authors' contributions
PB has compiled information about the systems and has designed the benchmarks together with DM and VM. DM and VM have implemented the benchmarks. DM, VM, and BHP have conducted the benchmarks. All authors read and approved the final manuscript.

Funding
Open Access funding enabled and organized by Projekt DEAL. This research has been conducted as part of EU H2020 EarthServer-2 and German BigDataCube.

Availability of data and materials
The benchmark code is available as part of the rasdaman source code [94]. The source/executable code of the systems benchmarked is available at the URLs given in the text; its contents is not under the control of the article authors.

Competing interests
The team has conducted this research based on their experience with implementing an Array DBMS, rasdaman. The authors are also employed at research spin-off rasdaman GmbH.

Annex 1: Array DBMS conceptual 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 variable 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, shown 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
Horiz. 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

	Array DBMS						
	Full-stack Array DBMS				**Add-on array support**		
	rasdaman	**SciDB**	**SciQL**	**EXTASCID**	**PostGIS Raster**	**Oracle GeoRaster**	**Teradata Arrays**
Processing model							
Query language expressiveness (built-in)	declarative array QL	declarative array QL	declarative array QL	No, function calls	Array functions with specific microsyntax, nottightly 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 integrated 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							

	Array DBMS						
	Full-stack Array DBMS				**Add-on array support**		
	rasdaman	**SciDB**	**SciQL**	**EXTASCID**	**PostGIS Raster**	**Oracle GeoRaster**	**Teradata Arrays**
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, including 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 language interfaces							
Domain-independent	C++, Java, python, R, JavaScript	C++, python, R, julia	Java, perl, R, Ruby, PHP, python, R	?	C, PHP, python, Java	Java & all other languages supported by Oracle	C, COBOL, PL/1
Domain-specific	many geo clients via OGC standards: OpenLayers, QGIS, NASA WorldWind, ...	?	?	?	MapServer, GeoServer, Deegree, QGIS, ...	?	No
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 functionality only on 1D arrays; array size limited to less than 64 kB, array generation to 2559 cells; array operators in function syntax, no infix (like "a + b");

	Array tools									
	OPeNDAP Hyrax	**xarray**	**TensorFlow**	**Wendelin .core**	**Google Earth Engine**	**OpenData 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	Llikely various numeric types	NetCDF cell data types	C++ data types	C++ data types	C primitives?	Numeric types, fixed array, variable array, string
Null values	No	Yes	Yes (placeholders)	No	No	No	No	No	?	Yes
Data integration										
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	Astronomy	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	

	Array tools									
	OPeNDAP Hyrax	**xarray**	**TensorFlow**	**Wendelin .core**	**Google Earth Engine**	**OpenData Cube**	**xtensor**	**Boost:: geometry**	**Ophidia**	**TileDB**
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-side python	yes	In-memory	yes	no
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 tools									
	OPeNDAP Hyrax	**xarray**	**TensorFlow**	**Wendelin .core**	**Google Earth Engine**	**OpenData Cube**	**xtensor**	**Boost:: geometry**	**Ophidia**	**TileDB**
Array joins	No	Yes	Yes	No	Yes (function call)	No	yes	yes, main memory	yes, INTERCUBE operation; requires 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 invocation from within EE functions, but through own wrapping code in host language	No	Yes, via C++ user code	Yes, via C++ code	Yes, via shell or python	No
Import/export										
Data formats	Import: csv, dapreader, dsp, ff, fits, gdal, h5, hdf4/5, … Export: ascii, netCDF, Binary (DAP), xml	Large number of formats, anything that python can understand through a library	Export: binary checkpoint files (state) + SavedModel; 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 tools									
	OPeNDAP Hyrax	**xarray**	**TensorFlow**	**Wendelin .core**	**Google Earth Engine**	**OpenData Cube**	**xtensor**	**Boost:: geometry**	**Ophidia**	**TileDB**
Array cells update	No	Any cell or region	Any cell or region	Any cell or region	Down to single cell level	No update functionality	Any cell or region	Down to single cell	No update function-ality	Yes
Client language interfaces										
Domain-inde-pendent	C++, Java,	Python	Python, C, Java, go	Python, C, Fortran	JavaScript, python	Python	C++	C++	python	C++
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	
	SciHadoop	**SciSpark**
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

	MapReduce	
	SciHadoop	**SciSpark**
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 language 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

Annex 2: Array DBMS physical tuning 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 (100 × 100 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 elimination	Yes	?	?	No	No	No	No
Cost-based optimization	Yes	?	?	No	No	No	No
Just-in-time query compilation, mixed hardware	Yes	No	No	No	No	No	No

	Array tools									
	OPeNDAP	**xarray**	**Tensor Flow**	**wendelin. core**	**Google Earth Engine**	**Open Data Cube**	**xtensor**	**boost:: geom etry**	**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 underlying 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 hard ware	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

Annex 3: Array DBMS architectural features 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 reassembled through query)	2D, regular	No
Non-regular tiling	Any nD tiling	No	No	Yes	yes (with manual reassembly 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 preexisting 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 and 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 100 × 100		Array size limited to less than 64 kB

	Array tools									
	OPeNDAP	**xarray**	**TensorFlow**	**wendelin.core**	**Google Earth Engine**	**OpenDataCube**	**xtensor**	**boost::geometry**	**Ophidia**	**TileDB**
Architecture paradigm	Web frontend, based on DAP protocol, with format-specific processors in the background	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-dimensional arrays
Storage organization										
Partitioning	Yes, as per NetCDF	No (main memory centric)	No	Yes, via NEO ZODB, but array agnostic	Yes (typically, 256 × 256 pixels to match input preprocessing)	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 ingestion script	No	No	No	Yes
Tiles on separate computers	No	No	No	Yes, via NEO ZODB	Yes	No	No	No	No	Yes, via VFS (virtual file system) with distribution similar to HDFS
Processing on preexisting archives (with their individual organization)	No	No	No	No	No (data must sit in Google)	No	No	No	No	No

	Array tools									
	OPeNDAP	**xarray**	**TensorFlow**	**wendelin.core**	**Google Earth Engine**	**OpenDataCube**	**xtensor**	**boost::geometry**	**Ophidia**	**TileDB**
Tape archives	No	No	No	No	No	No	No	No	No	No
Processing and 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 and 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)

Received: 26 July 2020 Accepted: 12 December 2020
Published online: 02 February 2021

References

1. Abadi M, et al. Tensorflow: A system for large-scale machine learning. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). 2016. p. 265–283.
2. Abadi D. On Big Data, analytics and hadoop. ODBMS Industry Watch. 2012. http://www.odbms.org/blog/2012/12/on-big-data-analytics-and-hadoop-interview-with-daniel-abadi/. Accessed 23 Aug 2020.
3. Abadi M. TensorFlow: Learning functions at scale. Proc. ACM SIGPLAN Intl. Conference on Functional Programming. St Petersburg, USA, 2016.
4. Andrejev A, Baumann P, Misev D, Risch T. Spatio-temporal gridded data processing on the semantic web. Proc. Intl. Conf. on Data Science and Data Intensive Systems (DSDIS). Sydney, Australia, 2015.
5. Baumann P. A database array algebra for spatio-temporal data and beyond. Proc. Intl. Workshop on Next Generation Information Technologies and Systems (NGITS). Zikhron Yaakov, Israel. Springer LNCS 1649. 1999.
6. Baumann P. Array Databases. In: Özsu T, Liu L, editors. Encyclopedia of Database Systems. Springer, 2017.
7. Baumann P. Beyond Rasters: Introducing The New OGC Web Coverage Service 2.0. Proc. ACM SIGSPATIAL GIS. San Jose, USA, 2010.
8. Baumann P, Feyzabadi S, Jucovschi C. Putting pixels in place: a storage layout language for scientific data. Proc. IEEE ICDM Workshop on spatial and spatiotemporal data mining (SSTDM). Sydney. 2010;194:201.
9. Baumann P, Hirschorn E, Maso J, Dumitru A, Merticariu V. Taming Twisted Cubes. Proc. ACM SIGMOD Workshop on Managing and Mining Enriched Geo-Spatial Data (GeoRich). San Francisco. 2016.
10. Baumann P, Hirschorn E, Maso J. OGC Coverage Implementation Schema 1.1. OGC document 09-146r8. http://docs.opengeospatial.org/is/09-146r8/09-146r8.html. Accessed 23 Aug 2020.
11. Baumann P, Hirschorn E, Maso J, Merticariu V, Misev D. All in One: Encoding Spatio-Temporal Big Data in XML, JSON, and RDF without Information Loss. Proc. IEEE Intl. Workshop on Big Spatial Data (BSD). Boston, 2017.
12. Baumann P, Holsten S. A Comparative Analysis of Array Models for Databases. Proc. Database Theory and Application (DTA). Jeju Island, Korea. 2011, Communications in Computer and Information Science 258, Springer 2011.
13. Baumann P, Howe B, Orsborn K, Stefanova S: Proc. EDBT/ICDT Workshop on Array Databases. Uppsala, Sweden, 2011. https://www.rasdaman.com/ArrayDatabases-Workshop/. Accessed 23 Aug 2020.
14. Baumann P. Language Support for Raster Image Manipulation in Databases. Proc. Int. Workshop on Graphics Modeling, Visualization in Science & Technology. Darmstadt, Germany, 1992.
15. Baumann P, Merticariu V. On the efficient evaluation of array joins. Proc. IEEE Big Data Workshop Big Data in the geo sciences. Santa Clara; 2015.
16. Baumann P, Misev D, Merticariu V, Pham Huu B, Bell B, Kuo KS. Array Databases: Concepts, Standards, Implementations. RDA Array Database Assessment Working Group. 2018, https://rd-alliance.org/system/files/Array-Databases_final-report.pdf. Accessed on 23 Aug 2020.
17. Baumann P. OGC Web Coverage Processing Service (WCPS) Language interface standard, version 1.0. OGC document 08-068r2. https://www.ogc.org/standards/wcps. Accessed 23 Aug 2020.
18. Baumann P. On the management of multidimensional discrete data. VLDB J. 1994;4(3):401: – 444.
19. Baumann P, Rossi AP, Bell B, Clements O, Evans B, Hoenig H, Hogan P, Kakaletris G, Koltsida P, Mantovani S, Marco Figuera R, Merticariu V, Misev D, Pham Huu B, Siemen S, Wagemann J: Fostering cross-disciplinary earth science through datacube analytics. In: Mathieu PP, Aubrecht C, editors. Earth observation open science and innovation - changing the world one pixel at a time. International Space Science Institute (ISSI), 2017; 91:119.
20. Baumann P, Stamerjohanns H. Towards a systematic benchmark for array database systems. Proc. Workshop on Big Data Benchmarking (WBDB). Pune. 2021. Springer LNCS 8163.
21. Baumann P. The Datacube Manifesto. http://earthserver.eu/tech/datacube-manifesto. Accessed 23 Aug 2020.
22. Baumann P. The OGC Web Coverage Processing Service (WCPS) Standard. Geoinformatica. 2010;14(4):447:479.
23. Big Earth. Datacube Standards – Understanding the OGC/ISO Coverage Data and Service Model. http://standards.rasdaman.com. Accessed 23 Aug 2020.
24. Big Earth Datacube Standards. https://standards.rasdaman.com. Accessed 23 Aug 2020.
25. Blaschka M, Sapia C, Höfling G, Dinter B. Finding your way through multidimensional data models. Proc. DEXA Workshop Data Warehouse Design and OLAP Technology (DWDOT). Vienna. 1998;198:203.
26. Boost. boost. http://www.boost.org. Accessed 23 Aug 2020.
27. Boost. boost. https://github.com/boostorg/boost. Accessed 23 Aug 2020.
28. Brodie M, Blaustein B, Dayal U, Manola F, Rosenthal A. CAD/CAM database management. IEEE Database Eng Bull. 1984;7(2):20.
29. Bekla J, et al. XLDB 2018. https://conf.slac.stanford.edu/xldb2018/agenda. Accessed 28 Sep 2020.
30. Brown PG. Overview of SciDB: large scale array storage, processing and analysis. Proc. ACM SIGMOD. Indianapolis. 2010; 963:968.
31. Buck J. SciHadoop. https://github.com/four2five/SciHadoop. Accessed 23 Aug 2020.
32. Cheng Y, Rusu F. Astronomical data processing in EXTASCID. Proc. Intl. Conf. on Scientific and Statistical Database Management (SSDBM). Baltimore. 2013; 1:4.
33. Cheng Y, Rusu F. Formal representation of the SS-DB benchmark and experimental evaluation in EXTASCID. Distributed Parallel Databases. 2015;277:317.
34. Codd EF. A relational model of data for large shared data banks. Comm ACM. 1970;13(6):377:387.

35. CODE-DE Datacubes. https://processing.code-de.org/rasdaman. Accessed 23 Aug 2020.
36. Cudre-Maroux P, et al. SS-DB: a standard science DBMS benchmark. 2010. (submitted for publication).
37. Cudre-Mauroux P, et al. A demonstration of SciDB: a science-oriented DBMS. VLDB. 2009;2(2):1537.
38. Dean J, Ghemawat S. MapReduce. Simplified data processing on large clusters. Proc. 6th Symposium on Operating System Design and Implementation (OSDI), San Francisco. 2004. USENIX Association 2004. p. 137–150.
39. Dehmel A. A Compression engine for multidimensional array database systems. PhD Thesis, TU München. 2002.
40. Dumitru A, Merticariu V, Baumann P. Exploring cloud opportunities from an array database perspective. Proc ACM SIGMOD Workshop on Data Analytics in the Cloud (DanaC). Snowbird. 2014.
41. EarthServer Coverage Webinars. https://earthserver.xyz/wcs. Accessed 23 Aug 2020.
42. Ensor P. Organizational renewal —tearing down the functional silos. AME Target: Summer; 1988. p. 4–16.
43. Furtado P, Baumann P. Storage of multidimensional arrays based on arbitrary tiling. Proc. Intl. Conference on Data Engineering (ICDE). Sydney. 1999.
44. GeoTrellis. GeoTrellis. http://geotrellis.io. Accessed 23 Aug 2020.
45. GeoTrellis: GeoTrellis. https://github.com/geotrellis. Accessed 23 Aug 2020.
46. Gibson W. Data, data everywhere – the economist special report: managing information. 2010. http://www.economist.com/node/15557443. Accessed 23 Aug 2020.
47. Google. E, Engine. https://earthengine.google.com. Accessed 23 Aug 2020.
48. Gorelick N, Hancher M, Dixon M, Ilyushchenko S, Thau D, Moore R. Google earth engine: planetary-scale geospatial analysis for everyone. Remote Sens Environ. 2017;202:27.
49. Gutierrez C, Hurtado C, Mendelzon A. Formal aspects of querying RDF databases. Intl. Workshop on Semantic Web and Databases. Co-located with VLDB 2003, Humboldt-Universität, Berlin. 2003, p. 293–307.
50. Guttman A. R-Trees: a dynamic index structure for spatial searching. Proc. ACM SIGMOD. 1984;47:57.
51. Hey T, Tansley S, Tolle K. The fourth paradigm. Microsoft research, October 2009, http://research.microsoft.com/en-us/collaboration/fourthparadigm/. Accessed 23 Aug 2020.
52. Howard T. A Shareable centralised database of KRT3 - a hierarchical graphics system based on PHIGS. Proc. Eurographics 1987, Eurographics Association, 1987.
53. Indyk P. Nearest neighbours in high-dimensional spaces. In: Goodman JE, O'Rourke J, editors. Handbook of discrete and computational geometry. London: Chapman and Hall; 2004. p. 877–892.
54. INSPIRE coverage download services. http://inspire.ec.europa.eu/id/document/tg/download-wcs. Accessed 23 Aug 2020.
55. ISO. 19123-1:2019 Coverage Fundamentals (Working Draft). http://external.opengeospatial.org/twiki_public/CoveragesDWG/WebHome#Related_Standards. Accessed 23 Aug 2020.
56. ISO. Information technology—Database languages—SQL—Part 15: Multi-Dimensional Arrays (SQL/MDA). ISO IS 9075-15:2017. https://www.iso.org/standard/67382.html. Accessed 23 Aug 2020.
57. ISO. Information technology—Database languages—SQL—Part 1: Framework (SQL/Framework). ISO IS 9075-1:2016.
58. Ivanova M, Kersten ML, Manegold S. Data vaults: a symbiosis between Database technology and scientific file repositories. Proc. Intl. Conference on Scientific and Statistical Database Management (SSDBM). Athens. 2012; 485:494.
59. Iverson KE. A Programming Language. Wiley: New York. 1962.
60. Kaur K, Rani R, Modeling and querying data in NoSQL databases. Proc. IEEE Intl. Conf. on Big Data, Silicon Valley. 2013. p. 1–7.
61. Koubarakis M, Datcu M, Kontoes C, Di Giammatteo U, Manegold S, Klien E. TELEIOS: a database-powered virtual earth observatory. VLDB 5, 2012; 2010:2013.
62. Liakos P, Koltsida P, Kakaletris G, Baumann P. xWCPS: Bridging the gap between array and semi-structured data. Proc. Intl. Conference on Knowledge Engineering and Knowledge Management. Springer. 2015.
63. Liakos P, Koltsida P, Baumann P, Ioannidis Y, Delis A. A distributed infrastructure for earth-science big data retrieval. Intl J Cooperative Inform Syst. 2015;24(2): 1550002.
64. Liaukevich V, Misev D, Baumann P, Merticariu V. Location and processing aware datacube caching. Proc. Intl. Conference on Scientific and Statistical Database Management (SSDBM). New York. 2017. Article 34.
65. Marek-Spartz M. Comparing map algebra implementations for Python: Rasterio and ArcPy. Volume 18, Papers in Resource Analysis. Saint Mary's University of Minnesota Central Services Press. http://www.gis.smumn.edu/GradProjects/Marek-SpartzM.pdf. Accessed 23 Aug 2020.
66. Merticariu G, Misev D, Baumann P. Measuring storage access performance in array databases. Proc. Workshop on Big Data Benchmarking (WBDB). New Delhi. 2015.
67. Misev D, Baumann P. Enhancing Science Support in SQL. Proc. IEEE Big Data Workshop on Data and Computational Science Technologies for Earth Science Research. Santa Clara. 2015.
68. Misev D, Baumann P. Homogenizing data and metadata retrieval in scientific applications. Proc. ACM CIKM DOLAP. Melbourne. 2015; 25:34.
69. Misev BP. The Open-Source rasdaman Array DBMS. Proc. VLDB Workshop Big Data Open Source Systems (BOSS). New Delhi, India. 2016.
70. MonetDB: SciQL. https://projects.cwi.nl/scilens/content/platform.html. Accessed 23 Aug 2020.
71. MrGeo. MrGeo. https://github.com/ngageoint/mrgeo.. Accessed 23 Aug 2020.
72. Mundi Datacubes. https://mundi.rasdaman.com. Accessed 23 Aug 2020.
73. N.n. Hadoop. http://hadoop.apache.org/. Accessed 23 Aug 2020.
74. N.n. Spark. http://spark.apache.org/. Accessed 23 Aug 2020.
75. ODC. Open Data Cube. https://www.opendatacube.org. Accessed 23 Aug 2020.

76. ODC. Open Data Cube. https://github.com/opendatacube. Accessed 23 Aug 2020.
77. OGC Web Coverage Service. www.ogc.org/standards/wcs. Accessed 23 Aug 2020.
78. OGC Web Coverage Processing Service. www.ogc.org/standards/wcps. Accessed 23 Aug 2020.
79. OGC Compliance Testing. https://www.ogc.org/compliance. Accessed 23 Aug 2020.
80. Oosthoek J, Rossi AP, Baumann P, Misev D, Campalani P. PlanetServer: towards online analysis of planetary data. Planetary Data. 2012.
81. OPenDAP. http://www.opendap.org. Accessed 23 Aug 2020.
82. OPeNDAP. Software. https://www.opendap.org/software. Accessed 23 Aug 2020.
83. Ophidia: Ophidia. http://ophidia.cmcc.it. Accessed 23 Aug 2020.
84. Ophidia: Ophidia. https://github.com/OphidiaBigData. Accessed 23 Aug 2020.
85. Oracle: GeoRaster. http://docs.oracle.com/cd/B19306_01/appdev.102/b14254/geor_intro.htm. Accessed 23 Aug 2020.
86. Paradigm4: SciDB. https://www.paradigm4.com. Accessed 23 Aug 2020.
87. Paradigm4. SciDB Licensing. https://www.paradigm4.com/about/licensing/. Accessed 23 Aug 2020.
88. Paradigm4. SciDB Source Code. https://drive.google.com/drive/folders/0BzNaZtoQsmy2aGNoaV9Kdk5YZEE. Accessed 23 Aug 2020.
89. PlanetServer. http://planetserver.eu. Accessed 23 Aug 2020.
90. Planthaber G, Stonebraker M, Frew J. EarthDB: scalable analysis of MODIS data using SciDB. Proc. ACM SIGSPATIAL Intl. Workshop on Analytics for Big Geospatial Data. 2012. p. 11–19.
91. PostGIS. PostGIS Developers Wiki. https://trac.osgeo.org/postgis/wiki/DevWikiMain. Accessed 23 Aug 2020.
92. PostGIS. PostGIS Raster manual. http://postgis.net/docs/manual-dev/using_raster_dataman.html. Accessed 23 Aug 2020.
93. PostGIS. Raster PostGIS. https://postgis.net/docs/using_raster_dataman.htm. Accessed 23 Aug 2020.
94. Rasdaman. https://www.rasdaman.org. Accessed 23 Aug 2020.
95. Rasdaman. rasdaman. https://rasdaman.com. Accessed 23 Aug 2020.
96. Reiner B, Hahn K: Hierarchical Storage Support and Management for Large-Scale Multidimensional Array Database Management Systems. Proc. DEXA. Aix en Provence, France, 2002.
97. Ritter G, Wilson J, Davidson J. Image Algebra: An Overview. Computer Vision, Graphics, and Image Processing. 49(1)1990;297:331.
98. Rusu F, Cheng Y. A survey on array storage, query languages, and systems. arXiv preprint arXiv:1302.0103, 2013.
99. Rusu F. EXTASCID. http://faculty.ucmerced.edu/frusu/Projects/GLADE/extascid.html. Accessed 123 Aug 2020.
100. Sarawagi S, Stonebraker M: Efficient organization of large multidimensional arrays. Proc. Intl. Conf. on Data Engineering (ICDE). Houston. 1994. p. 328–336.
101. SciSpark. https://scispark.jpl.nasa.gov. Accessed 23 Aug 2020.
102. SciSpark: SciSpark. https://github.com/SciSpark. Accessed 23 Aug 2020.
103. Session ESSI2.2: Data cubes of Big Earth Data - a new paradigm for accessing and processing Earth Science Data. https://meetingorganizer.copernicus.org/EGU2018/posters/28035. Accessed 23 Aug 2020.
104. Soroush E, Balazinska M, Wang D. ArrayStore: a storage manager for complex parallel array processing. Proc. ACM SIGMOD. Athens. 2011. p. 253–264.
105. Soussi R, Aufaure MA, Zghal HB. Towards social network extraction using a graph database. In: Laux F, Strömbäck L, editors Intl. Conf. on Advances in Databases, Knowledge, and Data Applications (DBKDA), Menuires. 2010, p. 28–34.
106. Stancu-Mara S, Baumann PA, Comparative Benchmark of Large Objects in Relational Databases. Proc. IDEAS 2008, Coimbra, Portugal, November 2008.
107. Stonebraker M, Brown P, Zhang D, BeclaJ. SciDB: a database management system for applications with complex analytics. Comput Sci Eng. 2013;15(3):62.
108. Stonebraker M, Ugur C. "One Size Fits All": an idea whose time has come and gone. Proc. Intl. Conf. on Data Engineering (ICDE). Washington. 2005. p. 2–11.
109. Tan Z, Yue P, A comparative analysis to the array database technology and its use in flexible VCI derivation. Fifth Intl. Conference on Agro-Geoinformatics, July 2016, p. 1–5.
110. TensorFlow. TensorFlow Installation. https://www.tensorflow.org/install. Accessed 23 Aug 2020.
111. Teradata. Multidimensional array options. https://docs.teradata.com/reader/eWpPpcMoLGQcZEoyt5AjEg/0BjYNH4d7gS8CrkifJWErg. Accessed 23 Aug 2020.
112. 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. Accessed 23 Aug 2020.
113. The RDF Data cube vocabulary. https://www.w3.org/TR/vocab-data-cube/. Accessed 23 Aug 2020.
114. The EarthServer. Datacube federation. earthserver.xyz. Accessed 23 Aug 2020.
115. TileDB: TileDB. https://github.com/TileDB-Inc. Accessed 23 Aug 2020.
116. TileDB. TileDB. https://tiledb.io. Accessed 23 Aug 2020.
117. Tomlin D. A Map Algebra. Harvard Graduate School of Design, 1990.
118. W3C. Extensible Markup Language (XML) 1.0. https://www.w3.org/TR/REC-xml/. Accessed 23 Aug 2020.
119. Webster P. Supercomputing the climate: NASA's Big Data Mission. CSC World Computer Sciences Corporation, 2012.
120. Wendelin.core. Wendelin.core. https://lab.nexedi.com/nexedi/wendelin.core. Accessed 23 Aug 2020.
121. Wendelin.core. Licensing. https://www.nexedi.com/licensing. Accessed 23 Aug 2020.
122. Wu J. ArrayUDF Explores structural locality for faster scientific analyses. Proc. XLDB, Stanford. 2018.
123. Xarray. xarray. http://xarray.pydata.org. Accessed 23 Aug 2020.

124. Xtensor. xtensor. http://quantstack.net/xtensor. Accessed 23 Aug 2020.
125. Xtensor. xtensor. https://github.com/QuantStack/xtensor. Accessed 23 Aug 2020.
126. Zhang Y, Kersten ML, Ivanova M, Nes N. SciQL, bridging the gap between science and relational DBMS. Proc. IDEAS. 2011, p. 124–133.

Publisher's Note