## FP7- Grant Agreement no. 283393 – RadioNet3

Project name: Advanced Radio Astronomy in Europe

Funding scheme: Combination of CP & CSA

Start date: 01 January 2012

Duration: 48 month



# Deliverable D10.10

Scientific publication on the results of the demonstrator, and the overall performance gains

Due date of deliverable: 2015-02-28

Actual submission date: 2015-07-01

Deliverable Leading Partner: EUROPEAN SOUTHERN OBSERVATORY – ESO EUROPEAN ORGANISATION FOR ASTRONOMICAL RESEARCH IN THE SOUTHERN HEMISPHERE (ESO)



# **1** Document information

Туре	Demonstrator
Title	Scientific publication on the results of the demonstrator, and the overall performance gains
WP	10 (Hilado)
Authors	G.N.J. van Diepen (ASTRON)

### 1.1 Dissemination Level

Dissemination Level					
PU	Public	х			
PP	Restricted to other programme participants (including the Commission Services)				
RE	Restricted to a group specified by the consortium (including the Commission Services)				
со	Confidential, only for members of the consortium (including the Commission Services)				

### 1.2 Content

1	Do	cument information	2
	1.1	Dissemination Level	2
	1.2	Content	3
2	Sho	ort description	4

### 2 Short description

Many changes have been made in Casacore and CASA code to improve performance, especially on multi-core systems and on distributed platforms using a large global file system like Lustre. The two Casacore forks (Googlecode and CASA) have been unified; it is now available at https://github.com/casacore/casacore.

The attached publication discussing the Casacore Table Data System has been accepted by the Astronomy & Computing journal for their special issue about astronomical data formats. It discusses several of the improvements made in this Joint Research Activity-Hilado of the RadioNet3 program and shows performance. The paper "Casacore Table Data System and its use in the MeasurementSet" is attached to this document.

Several other changes, not discussed in the paper, are given below.

- Made all statics in casacore thread-safe, so it can be used safely in a multi-threaded environment.
- Multi-threaded sorting and other algorithms to improve performance.
- Improvements in handling of coordinates and FITS.
- Addition of arbitrary attributes and multiple beams to images.
- Made sure all code compiles without warnings and all tests pass, also when using a checking tool like valgrind.

### Copyright

© Copyright 2015 RadioNet3

This document has been produced within the scope of the RadioNet3 Projects. The utilization and release of this document is subject to the conditions of the contract within the 7<sup>th</sup> Framework Programme, contract no, 283393

### ARTICLE IN PRESS

Astronomy and Computing **(1111**) **111**-**111** 



Contents lists available at ScienceDirect

### Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

### Full length article

## Casacore Table Data System and its use in the MeasurementSet\*

### G.N.J. van Diepen

ASTRON, Oude Hoogeveensedijk 4, 7991PD Dwingeloo, Netherlands

#### ARTICLE INFO

Article history: Received 14 March 2015 Received in revised form 10 June 2015 Accepted 10 June 2015 Available online xxxx

Keywords: Astronomy Storage Query language Data format

#### ABSTRACT

The Casacore Table Data System (CTDS) has been developed as part of the CASA (formerly AIPS++) package for (radio-)astronomical data processing. It offers a relational-like data model extended by the ability to store large arrays. A versatile query language can be used for data queries and manipulation. CTDS forms the basis of the MeasurementSet, the primary data format used for the data acquisition and data processing at various radio telescopes (e.g., WSRT, LOFAR, ASKAP, JVLA, ALMA). This paper discusses the most important features of CTDS, its performance, and how it is used. Also comparisons with other data formats are made.

© 2015 Elsevier B.V. All rights reserved.

Astronomy

#### 1. Introduction

The Casacore Table Data System has existed for over 20 years, but there has been no paper describing it. Recent developments, and its use by large radio telescopes producing TBytes of data, motivate production now of a publication, whose description of lessons learned can make a timely contribution to the current debate within the wider community regarding astronomical data formats.

In 1992 it was decided that the Astronomical Image Processing System (AIPS) should have a successor, to be called AIPS++. A few institutes teamed up and started the analysis and design in 1992. Over time AIPS++ was renamed to CASA (Common Astronomy Software Applications). In 2006 it was decided to split off Casacore, the core part of CASA, to differentiate between the commonly applicable software and the software developed for the ALMA and JVLA instruments. Casacore is now used at several radio telescopes around the world.

Farris analysed the needs for the storage and processing of future radio-astronomical datasets. The results are summarised in the description of the first CTDS version (van Diepen and Farris, 1993). The analysis showed that various applications have different access patterns to the data, requiring flexible data access. For ad hoc applications like data inspection and plotting, flexible data selection is needed. Operations like calibration and imaging draw

<sup>†</sup> Part of the research leading to these results has received funding from the European Commission Seventh Framework Programme (FP/2007–2013) under grant agreement No. 283393 (RadioNet3).

E-mail address: diepen@astron.nl.

http://dx.doi.org/10.1016/j.ascom.2015.06.002 2213-1337/© 2015 Elsevier B.V. All rights reserved. on large bodies of data. Multi-dimensional arrays are the most natural modes for such scientific data. It was concluded that the data system will play a key role for efficient and flexible data access to make the implementation of existing and new algorithms easily possible. The relational data model (Codd, 1970) was quite well suited, but needed to be extended by the ability of storing arrays.

In 1993 the Flexible Image Transport System (FITS) (Wells et al., 1981) was quite basic and not well suited for interferometric visibility data. The new version of the Hierarchical Data Format (HDF5<sup>1</sup>) was in its early development stages, while other data formats did not exist. Therefore it was decided to design a data system, the Casacore Table Data System (CTDS), meeting the requirements following from the analysis. The first development took place in 1993 and since then it has seen many improvements. It can be seen as a NoSQL<sup>2</sup> system building on the relational model with two important features:

- The value in a table cell can be a multi-dimensional array.
- There are no explicit primary keys. Instead the row number is the implicit key in a table. This is possible because in principle rows are never deleted from a table.

A particular example of a Casacore table is the MeasurementSet containing the observed visibility data. It is briefly described in Section 2.1.

CTDS was developed in an era where parallel processing was hardly present.<sup>3</sup> This appeared not to be a real problem for the

<sup>1</sup> http://www.hdfgroup.org.

Please cite this article in press as: van Diepen, G.N.J., Casacore Table Data System and its use in the MeasurementSet. Astronomy and Computing (2015), http://dx.doi.org/10.1016/j.ascom.2015.06.002

<sup>&</sup>lt;sup>2</sup> Not only SQL; see en.wikipedia.org/wiki/NoSQL.

 $<sup>^{3}</sup>$  A collection of desktops could be seen as an unsupervised distributed processing system.

### ARTICLE IN PRESS

#### G.N.J. van Diepen / Astronomy and Computing 🛚 ( 💵 💷 )

new large-scale SKA pathfinder instruments because their data can be partitioned by spectral band. Processing steps like flagging and calibration can be done independently for each partition making embarrassingly parallel processing on a cheap cluster with local disks possible. Such a system has the benefit that IO scales linearly (an extra node means extra IO capacity). Another benefit is that processing can be brought to the data, not the other way around. This scheme is used by LOFAR to handle observations consisting of several TBytes of data. Typically, 384 spectral bands are distributed over 96 nodes. It is being considered for handling the Square Kilometre Array (SKA) data. The virtual concatenation mechanism of CTDS makes it possible to treat all those parts as a single MeasurementSet which is exploited by the CASA package for the JVLA and ALMA data processing. Although CTDS does not support parallel IO natively, it is quite adaptable. Section 5.4 discusses recent developments making parallel IO possible.

CTDS has a strong distinction between the logical and the physical data models described in Sections 2 and 3. The user's view of a table is the logical data model, a simple collection of rows and columns. Relations with other tables are defined in columns containing foreign keys, usually a row number. The logical data model is mapped to the underlying physical data model, where data managers take care of storing and retrieving the data in an efficient and flexible way. It makes it quite easy to develop specialised and experimental storage methods by developing a new data manager.

CTDS comes with a powerful SQL-like query language, TaQL, making arbitrary data selection, sort, and manipulation possible. The result of a selection and sort is a view on the original table, behaving as an ordinary table.

The Casacore Table Data System is written in C++ as part of the Casacore<sup>4</sup> software package and makes use of other parts of Casacore. The low-level C++ interface can be used by applications, that have to link against the Casacore libraries. The Pyrap<sup>5</sup> package offers a high-level interface in Python. Astronomers use the Python interface quite often for ad hoc data plotting and manipulation.

#### 2. Logical data model

Detailed storage parameters should be hidden as much as possible for the regular user and application programmer. CTDS has been designed that way by using a logical data model as a simple view on the data. Similar to a relational database table the data in a CTDS table are logically organised in rows and columns. Columns containing keys define the relations between tables. As described in Section 3, data managers take care of storing and retrieving the data.

Each cell in a column can contain a scalar value or a multidimensional array. Individual cells can be accessed and an entire column can be accessed as an (N + 1)-dim array. Access to array slices and column slices make flexible data access possible. The schema (table description) defines the data type of a column and optionally the dimensionality or shape of arrays to be stored in a column. All basic data types (boolean, integer, float, complex, and fixed and variable length string) are natively supported. Compound data types (possibly nested) are supported by aggregating the data in a struct-like data type. Similar to FITS (Pence et al., 2010) a CTDS table can have keywords to define some properties of the table or its columns. They are used for a few purposes:

- References to subtables containing the metadata of a dataset. They define the relations between the main table and its subtables. The MeasurementSet (Kemball and Wieringa, 2000), described in Section 2.1, is a good example how subtables are used.
- The definition of the physical unit of the data in a column. The TaQL query language uses it to do unit conversion if needed.
- The definition of the reference frame of coordinates in a column. For example, J2000 for celestial coordinates or ITRF for earth positions. They are directly related to the Measures<sup>6</sup> module in Casacore that is used for coordinate conversions.

CTDS tables do not have the concept of a primary key. Instead the row number in a table serves as the virtual key avoiding the need of indices and key lookup. This is similar to array indices used in SciQL (Zhang et al., 2012). Using the row number is adequate because the tables are usually write-once. It is possible to remove rows, but in practice it is never done because data selection is used to ignore rows. Another advantage of using the row number is that data in external files can be used directly as long as a row number can be mapped to a file offset. This is discussed in more detail in Section 3.2. Arbitrary views on a table or set of tables can be made using selection, sort, iteration, or concatenation. These operations result in a so-called reference table that behaves as an ordinary table, thus has the same logical data model and offers the same access functionality. The reference tables can be made persistent.

A CTDS table is represented as a directory containing several files. The name of a table is the name of the directory. The table schema and some auxiliary info are stored in a file created at the logical data model level. All other files are created and accessed at the physical data model level. A subtable is usually a subdirectory of the main table. It means that in the case of, for example, the MeasurementSet all data and metadata of an observation are in a single directory which can be moved, tarred, etc.

### 2.1. MeasurementSet

A particular example of a set of CTDS tables is the so-called MeasurementSet (Kemball and Wieringa, 2000). It is a structured collection of tables holding the interferometric visibility data and the metadata. It is used in several examples in other sections, hence a short description of the MeasurementSet is given.

Fig. 1 shows the tables and columns in a MeasurementSet. It also shows the relations between the tables. The main table contains the bulk data. Each row contains the visibility data matrix (usually 4 correlations and many frequencies) for a particular time, baseline, and possibly other parameters like field. Corresponding flags tell if a visibility has to be ignored. Metadata columns define the baseline, time, and other parameters of the visibility matrices. The most important columns with their data types, shapes, and units are given in Table 1. The MeasurementSet has additional subtables containing the metadata of the entire observation. For example, the ANTENNA subtable defines the name, position, and other properties of the telescope antennae. The FIELD subtable defines the phase centre of the observed field (usually given as J2000 right ascension and declination in radians). Metadata columns define the table relations. For example, column ANTENNA1 in the main table contains the row number in the ANTENNA subtable for the first antenna of a baseline.

The MeasurementSet format is fully extendible by adding instrument-specific subtables and columns. The MeasurementSet definition prescribes that such columns and subtables should have

<sup>&</sup>lt;sup>4</sup> https://github.com/casacore/casacore.

<sup>&</sup>lt;sup>5</sup> https://github.com/casacore/python-casacore.

<sup>6</sup> http://www.astron.nl/casacore/trunk/casacore/doc/html/group\_Measures\_ module.html.



G.N.J. van Diepen / Astronomy and Computing I (IIII) III-III



Fig. 1. Schema of the LOFAR MeasurementSet.

 Table 1

 Important columns in the MeasurementSet main table

Column	Format	Unit	Description		
TIME	Double	S	Integration midpoint		
INTERVAL	Double	S	Sampling interval		
ANTENNA1	Int		First antenna of baseline		
ANTENNA2	Int		Second antenna of baseline		
DATA_DESC_ID	Int		Data descriptor (spectral band)		
FIELD_ID	Int		Row number in FIELD subtable		
UVW	Double(3)	m	UVW coordinates of baseline		
DATA	Complex(nc,nf)		Visibilities per time, baseline		
WEIGHT	Float(nc)		Visibility weights		
FLAG	Bool(nc,nf)		True = visibility is bad		
FLAG_ROW	Bool		True $=$ entire row is bad		

the instrument's name as a prefix. Schoenmakers and Renting (2012) define the structure of the MeasurementSets used at the LOFAR telescope. Its structure is shown in Fig. 1. It has several LOFAR-specific columns and subtables, mostly used to describe the layout of the dipoles and tiles in the stations (de Vos et al., 2009), the LOFAR equivalent of radio-telescope antennae.

#### 3. Physical data model

The logical data model has to be mapped to disk in an efficient way, which is taken care of by the physical data model. Its data managers are responsible for storing/retrieving the data in the columns of the logical data model. The data manager best suited for handling a column depends on the type of data. Basically a dataset can contain two types of data:

• The bulk data. For example, the pixels in an image or the visibilities and flags in a MeasurementSet. These data are multi-dimensional arrays.

• The metadata describing the bulk data like the baselines and times. Data selection is usually based on the metadata, so having all metadata close together can result in much better query performance.

To achieve efficient storage and retrieval, the table designer needs to have some knowledge of the data manager properties. There are two basic types of data managers:

- A storage manager stores the data of columns in a disk file. CTDS comes with a few storage managers:
  - The Tiled Storage Manager is meant for the bulk data arrays. It stores them in a tiled (chunked) way for efficient access. It is described in more detail in Section 3.1.
  - The Standard and Incremental Storage Manager are meant for the metadata. The Incremental Storage Manager automatically compresses the data by only storing a value if different from the previous row. The Standard Storage Manager stores the data in a columnar way as used by modern database systems like MonetDB (Boncz et al., 2009). These data managers are very well suited for query purposes.

3

### ARTICLE IN PRESS

#### G.N.J. van Diepen / Astronomy and Computing 🛚 ( 💵 🖿 ) 💵 – 💵

Currently, each storage manager instance uses its own file(s) making the file structures quite easy (no special indices are needed). Usually this is not a problem, but in some cases the number of files can grow too large. A recent development has addressed this issue by combining many table files in a single file as described in Section 3.3.

• A virtual column engine calculates the data of a column on the fly. An example is the engine to compress 32-bit floating point values by scaling them to 16-bit integer values. Another example is an engine calculating the UVW coordinates on the fly instead of storing them.

Besides the predefined data managers mentioned above, specialised or experimental user-written data managers can be used. This feature makes CTDS very flexible and adaptive. More details and some examples are given in Section 3.2. At table creation time the columns are bound to data managers. Normally most metadata columns are bound to a single data manager instance to keep the number of files sufficiently low. It can also improve IO performance when a selection on multiple columns is done. Bulk data columns can be combined as well, but IO will be less efficient if only a single column needs to be accessed.

#### 3.1. Tiled storage manager

Tiling of data arrays is a technique used in several packages to improve IO performance when accessing data along different axes or when slices of an array are accessed. It also improves locality when accessing an area around a point in a multi-dimensional space. It is, for example, used in packages like HDF5<sup>7</sup> (where it is called chunking) and Karma.<sup>8</sup> Recently tiling has been added to FITS (White et al., 2012; Pence et al., 2012).

The basic idea is that storing a large array in a linear way makes access along the slower varying axes very slow. Instead the *N*-dim array is partitioned into smaller rectangular *N*-dim tiles. The data in each tile and the tiles are stored linearly. In this way access along any axis means that only a limited number of tiles are needed, but data have to be shuffled to map the tiles to the array in memory. The tile shape defines the efficiency of the access along the different axes. It should be defined such that the most frequently used access patterns are served best. When iterating through the data, the next data access often needs the same tiles. Therefore the recently accessed tiles should be kept in a cache.

CTDS supports the storage of data in a tiled way by means of the Tiled Storage Manager (TSM). For example, a spectral line image with axes Ra, Dec, Frequency and shape [8192,8192,4096] can be tiled like [128,128,64] to make access in all directions about equally fast. The TSM reads and writes a tile as a single IO block. It maintains a cache containing the most recently used tiles. The cache should be sized appropriately to minimise IO. If the application does not set the size explicitly, the TSM will guess it from the access pattern and resize if necessary. Usually it can guess well, but for more random-like access patterns the application should size the cache. Given the example image above, accessing all pixels along the Frequency axis for a given Ra, Dec needs a cache of 4096/64 tiles to avoid rereading tiles when iterating over Ra, Dec. Note this cache size assumes that iteration over Ra, Dec is not fully sequentially, but only sequentially per tile. Otherwise a larger cache would be required. The TSM keeps cache statistics, which can be used to learn if the cache behaves as expected. Section 3.4 shows some performance numbers of the Tiled Storage Manager compared to HDF5 and plain Linux IO.

#### 3.2. External data managers

Usually the predefined storage managers do a good job, but it can be desirable to write a dedicated or experimental storage manager. This is possible by writing a C++ class for such a storage manager and building it as a shared library. The class has to be derived from the abstract DataManager base class in CTDS. When CTDS detects that a table uses a data manager with an unknown name, it will dynamically load the shared library with that name and call a function to register the data manager. This feature is used to facilitate data access for the LOFAR and ALMA telescopes, detailed below.

- The LOFAR telescope can have a very high data acquisition rate (up to 16 GB/s) and direct disk IO is preferable. Therefore the visibility data coming from the correlator are stored in a LOFARspecific way. Yet, such data can be handled transparently as a CTDS table using the LofarStMan data manager developed for this purpose. It makes all applications fully agnostic to this specific data format. LofarStMan is a combination of a storage manager and virtual column engine. Only a few columns (visibility data and weights) are really stored, all others are derived from some header data or calculated on the fly.
- The ALMA telescope consortium chose to store and archive the data in the SDM format (Viallefond, 2006). A special storage manager was written to let the CASA<sup>9</sup> software access the visibility data directly in the SDM file instead of having to copy them all. For the ALMA data processing pipeline it resulted in a 20% performance gain and 50% reduction in disk usage.

#### 3.3. MultiFile

As discussed above CTDS can generate many files holding the data columns of a table. It proved to be problematic if many MeasurementSets were used jointly as done in the CASA package. Another disadvantage is that a parallel file system like Lustre (Schwan, 2003) cannot efficiently handle many small files.

The MultiFile mechanism has been developed to address this issue. It combines the data files in a single file in a way similar to the Sionlib (Frings et al., 2009) package. A simple index maps the data blocks in the MultiFile to the individual files. MultiFile adds the possibility of defining an IO block size matching the file system's block size for better performance. Tests have shown that the performance of MultiFile is comparable to the separate files currently used, so in the near future the MultiFile feature will probably become the default behaviour.

#### 3.4. Performance tests

Several tests have been done to compare the performance of CTDS, HDF5, and plain Linux IO. An uncompressed 32-bit float array with shape [1024,1024,1024] (4 GBytes) is written using different tile shapes. Such an array resembles an image cube with axes right ascension, declination, and frequency (the first axis varies most rapidly). The data were written tile by tile (thus sequentially) and retrieved using the following access patterns, for which the CTDS and HDF5 caches have been sized appropriately.

- Tile by tile (the optimal pattern). This pattern is useful for whole array operations like finding the maximum.
- Plane by plane (*xy*, *xz*, and *yz*). An image is often accessed in *xy* order, for example when displaying the image cube.

<sup>7</sup> http://www.hdfgroup.org.

<sup>&</sup>lt;sup>8</sup> http://www.atnf.csiro.au/computing/software/karma.

<sup>&</sup>lt;sup>9</sup> http://casa.nrao.edu.

#### G.N.J. van Diepen / Astronomy and Computing 🛚 ( 💵 💷 – 💵

Table 2		
Linux performance	tests on 4	GB dataset.

Tile shape	Access	CTDS	HDF5	Linux IO	CTDS	HDF5	
		Elapsed time			User tin	User time	
64,64,64	write	4.1	8.1	2.5			
32,32,32	write	3.8	8.2	2.4			
16,16,16	write	4.8	12.6	3.1			
8,8,8	write	11.5	45.2	4.5			
64,64,64	tile	1.9	1.3	1.4			
32,32,32	tile	1.4	1.4	1.3			
16,16,16	tile	1.5	4.6	1.3			
8,8,8	tile	6.4	26.5	1.5			
64,64,64	xy	2.5	3.5		1.6	2.3	
64,64,64	XZ	3.2	5.1		2.3	3.9	
64,64,64	уz	19.8	54.6		18.8	53.4	
64,64,64	x	3.2	73.4		2.6	72.6	
64,64,64	у	4.5	108.0		3.8	107.2	
64,64,64	z	14.3	127.6		13.3	126.6	
32,32,32	xy	2.5	8.7		1.7	7.5	
32,32,32	хz	3.4	11.8		2.6	10.6	
32,32,32	уz	16.1	51.6		14.9	50.2	
32,32,32	x	3.4	136.2		2.4	135.1	
32,32,32	у	4.7	177.2		3.6	176.2	
32,32,32	Ζ	12.2	196.3		11.3	195.2	

#### Table 3

Linux performance tests on 256 GB dataset.

Tile shape	Access	CTDS	HDF5	Linux IO
64,64,64	write	339.5	348.3	318.3
64,64,64	tile	371.1	372.0	343.2
64,64,64	xy	579.8	723.3	
64,64,64	x	537.5	>4200	

• Line by line (*x*, *y*, and *z*). An image is often accessed in *z* order for operations along the frequency axis like subtracting the continuum or rotation measure analysis.

The CTDS tile size and HDF5 chunk size define the IO block size. For comparison a dataset of 4 GBytes has been written and read using unbuffered IO (Unix read/write) with the same block sizes. In all tests no fsync has been done, because HDF5 does not have an fsync option. The tests have been performed on a Ubuntu system with 128 GB memory running Linux 3.2.0-61-generic. It has multiple Intel Xeon CPUs and cores, but only one core is used. Files are stored on a RAID6 system achieving approximately 800 MB/s. Table 2 shows the write and read results (in seconds) averaged over 5 runs. All tests are done with a tile shape of [64,64,64] and [32,32,32], while some tests are done with smaller tile shapes. The table shows the measured elapsed times and, where interesting, the time spent in user mode. The datasets used in these tests, fit in the Linux file cache, so hardly any disk access will be done. To take the cost of disk access into account, the elapsed times of tests using an array with shape [4096,4096,4096] (256 GB) are shown in Table 3. The Linux IO was measured with the dd command. The last HDF5 test was killed because it took too long.

It can be seen that the performance of CTDS and HDF5 is about the same when accessing the array data sequentially (by tile). It does not differ much from plain, unbuffered Linux IO. CTDS performs better than HDF5 when accessing the array by plane and especially by line. HDF5 spends a lot of time in user mode. Most likely the B-tree lookup done by HDF5 is the main culprit. HDF5 degrades quite severely when writing smaller chunks, probably because its B-tree gets much larger. The tables also show that access in *z* is significantly slower than access in *x* or *y*. This is probably caused by the large memory distance between subsequent *z* pixels causing bad memory/cache behaviour. Smaller tiles will alleviate this problem, which can already be seen from the fact that 32,32,32 requires less user time than 64,64,64. But small tiles will degrade IO, so an optimum has to be found. Note that Karma supports multilevel tiling to match the memory hierarchy.

#### 4. Table query language

Radio-astronomical data processing requires flexible data selection. It is common to select visibility data on baseline or spectral band, but selection on other criteria like the number of flagged visibilities is also very useful. CTDS comes with a query language (TaQL) (van Diepen, 2010) to make flexible data querying and manipulation possible. It consists of a large subset of SQL-92<sup>10</sup> with extensions to operate on arrays. It fully supports nested queries and aggregation (GROUPBY/HAVING clause). Its main properties different from SQL are:

- About any operand can be a scalar or an array. A set and a 1-dim array are interchangeable.
- A very rich set of mathematical, aggregation, and other functions. Many reduction functions can be applied to arrays. They result in a scalar if applied to the full array or result in a smaller array if applied for given axes only.
- Natural support of units and automatic conversion of units. The Astronomical Data Query Language (ADQL<sup>11</sup>) proposed by the International Virtual Observatory Alliance, has limited support of units by means of its IN\_UNIT function. Otherwise the author does not know of any other query language with unit support, while it appears to be a useful feature for scientific data processing.
- Specific operators and functions for cone searching (i.e., searching within a radius around a point at the sky).
- Join functionality is limited. Only joins using the row number are possible.
- The language can be extended by means of user defined functions (UDFs). They have to reside in a shared library that will be loaded dynamically. Two standard UDF libraries exist dealing with coordinate conversions (for sky directions, epochs, and earth positions) and dealing with the data in a MeasurementSet. The latter takes advantage of the knowledge of the MeasurementSet structure. In this way it is possible to get, say, the hourangle of the data in the MeasurementSet. This proved to be very useful for plotting purposes.

CTDS does not have persistent column indices, so TaQL is doing a sequential scan over a table. Indices are overkill for CTDS because creating and maintaining them is expensive, queries on a table are usually not very selective, and scanning a table is very fast and takes little IO because the metadata are stored closely together. For example, baseline selection on a MeasurementSet of 1.5 million rows takes less than 0.3 s on the system mentioned in Section 3.4. To get an impression of TaQL, some example queries are given below. A full description of the language can be found in its manual (van Diepen, 2010).

```
select gntrue(FLAG) from my.MS
select TIME,gntrue(FLAG) as NT from my.MS
groupby TIME having NT>0
```

counts the number of flagged visibilities in a MeasurementSet. The first command for the entire MS, the second command per time slot where it only selects the time slots having flagged data. Note this operation is quite fast because the standard CTDS storage managers compress boolean values to bits.

select from ref.ms t1, out.ms t2
where any(t1.DATA !~= t2.DATA)

Please cite this article in press as: van Diepen, G.N.J., Casacore Table Data System and its use in the MeasurementSet. Astronomy and Computing (2015), http://dx.doi.org/10.1016/j.ascom.2015.06.002

<sup>10</sup> http://en.wikipedia.org/wiki/SQL-92.

<sup>&</sup>lt;sup>11</sup> http://wiki.ivoa.net/twiki/bin/view/IVOA/ADOL.

### ARTICLE IN PRESS

#### 

can be used for regression testing. Operator = tests if the data are equal within floating point accuracy. Note this query is doing a equi-join on row number.

```
update my.ms set FLAG=False, FLAG_ROW=False
update my.ms set
```

FLAG=FLAG||ampl(DATA)>3\*median(ampl(DATA))

The first command clears all flags in a MeasurementSet.

The second command sets a flag where the data amplitude is too high. Note that the OR operator is used to keep the existing flags if already set.

update my.image set map = map - runningmedian(map,25,25)

subtracts the background noise from an image using the median in a [51,51] box around each pixel. For a larger image (tens of millions of pixels) this can become CPU-expensive.

#### 4.1. Selection and iteration

When processing data, it is very convenient to make crosscuts through the data, to combine data from different datasets, and to access the data in any order. CTDS has support for all of these. Crosscuts through the data can be made using a TaQL selection. Not only can a selection be done on rows, but it can also be done on a column containing arrays by taking a slice of the column. In this way one or more polarisations and/or frequency channels could be selected from a MeasurementSet.

A union of tables can be seen as a special form of selection. In SQL this is done by means of the UNION clause. CTDS can combine datasets by making a virtual concatenation of tables with the same schema. Such a union is a view on the original tables, thus very cheap to create. It behaves as an ordinary table, so any operation can be done on it. The CASA Multi-MeasurementSet idiom makes use of this feature to process the individual parts in parallel, while it is still possible to handle all data as a single table.

Accessing table rows in arbitrary order can be achieved using iteration, which can be seen as a specialised form of sorting and selection. It can be used to step through a table based on the contents of one or more columns. It is very similar to the GROUPBY clause in SQL, where each group is a subset of the table, thus a view on the table for the rows in that group. Iteration is very powerful and convenient. It is quite heavily used in the CASA package. Also ad hoc Python scripts use it quite often like:

import pyrap.tables as pt
t = pt.table ('my.ms')
for t1 in t.iter(['ANTENNA1','ANTENNA2']):
 data = t1.getcol('DATA')

This example opens a table and iterates through it on baseline. For each baseline it gets all data. However, iteration can be costly when accessing the bulk data. If the physical data order does not match the iteration order, it can result in low performance because of the many seeks involved. By tiling the data properly, this problem can be greatly reduced.

#### 5. Comparison with some other packages

Since the creation of CTDS, many other packages and database systems have been created. Some of these systems are briefly discussed and, where applicable, compared with CTDS.

#### 5.1. HDF5

HDF5 is a well-known package, developed at the end of the 1990's. It is used in many projects. It supports several storage methods like a table, group hierarchy, and big arrays.

A group hierarchy is the most commonly used method. MeasurementSet data could be stored as the hierarchy Band-Time-Baseline. A drawback of using a group hierarchy is the effort needed to access the data in another order. To access it in, say, Baseline-Time-Band order, specific code has to be written as no general method is available. Different access patterns are regularly needed in radio-astronomical data processing making this limitation a serious problem. For example, flagging is usually done per baseline for a (large) time-frequency window, while calibration requires all baselines per time/frequency. CTDS does not have this limitation as discussed in Section 4.1. Another drawback of using a group hierarchy in HDF5 is that selection on metadata other than the groups is impossible.

A second option is to store the data in HDF5 tables.<sup>12</sup> Selection on such a table can be done efficiently in Python using the PyTables<sup>13</sup> package, but only on columns containing scalar values. There is no query interface available in C or C++. A limitation of HDF5 tables is that the records in the tables are fixed length and can hold small arrays only.

A final option is to store all data in a single chunked array with axes time, baseline, frequency, and polarisation. HDF5 only supports selection by means of slicing, which makes selection on antenna or other parameters impossible. HDF5 chunk cache support is limited. It does not recognise access patterns, so sizing the cache is left to the user. Furthermore, resizing the cache can only be done after reopening the entire dataset. The performance tests in Section 3.4 show that HDF5's chunked data access can be far from optimal when iterating through a dataset.

HDF5 can support external files as long as the data layout is regular. Only a simple description of the file has to be passed to HDF5 to use this feature. CTDS can support any external file, but requires writing some code as discussed in Section 3.2. It is not particularly difficult, but extra effort is needed. It would be worthwhile to develop a CTDS storage manager that can handle a description file similar to HDF5.

A nice feature of HDF5 is its support of parallel IO, while CTDS does not support it. However, as discussed in Section 1 radioastronomical data processing is usually embarrassingly parallel, so parallel IO is hardly needed. In Section 5.4 it is shown that parallel IO can be added to CTDS using an external data manager.

Another feature of HDF5 is its standard support of chunk compression, but has the disadvantage that updating a compressed chunk wastes disk space if the chunk does not fit in the existing space.

#### 5.2. FITS

FITS has been around for about 30 years. It has grown over the years and recently tiling with compression has been added (White et al., 2012; Pence et al., 2012). The original FITS format was meant for image data. Later the binary table extension has been added to make it possible to store table data. These extensions are used to store visibility data, but there is no standard format. UVFITS (Greisen, 2012) and FITS-IDI (Greisen, 2008) are both used. In practice the FITS format is mostly used to transport the visibility data between data processing packages. The FITS data format is write-once and a file cannot be extended. Once created, it is not possible to add rows or columns to a binary table inside the FITS file. Also no tools exist to query a FITS table.

Price et al. (2014) compare FITS, HDF5, and MeasurementSet (thus CTDS). The authors mention there are compelling reasons

<sup>12</sup> https://www.hdfgroup.org/HDF5/Tutor/h5table.html.

<sup>13</sup> www.pytables.org.

to use HDF5, mainly its support of large data sets and a lossless compression ratio of 1.65x. As shown in this article the CTDS format also supports large datasets, but the lossless compression is an interesting feature. It might be worthwhile to add such a feature (as a data manager) to CTDS.

#### 5.3. SciDB

SciDB<sup>14</sup> is a recent database development aimed at distributed storage and processing of large amounts of data. It uses an Array Data Model, where the data are stored in a tiled way, possibly each tile on a different node. The tiles can overlap to avoid having to retrieve neighbouring data from another node if operations in a (small) region are done. An important property of SciDB is that data are never overwritten. An update results in a new array. It is too early to say if SciDB is suitable for the storage and processing of (radio-)astronomical data. It is required to handle ingest data rates of many GB/s (and much more for the SKA). The Large Synoptic Survey Telescope (LSST) was involved in the initial phase of SciDB, but is not anymore. SciDB and MonetDB tried to develop the ArrayQL<sup>15</sup> query language (based on SciQL), but no developments have taken place since 2012.

#### 5.4. ADIOS

The Adaptable IO System (ADIOS) (Liu et al., 2014), developed at Oak Ridge, is an advanced data storage package controlled by an XML file. It has full support of parallel IO and is very fast, but requires a lot of memory to achieve high performance. It caches as many data blocks as possible in memory to avoid disk accesses.

At ICRAR (Perth, Australia) ADIOS has been used in an experimental CTDS storage manager.<sup>16</sup> Tests showed its performance is comparable with the native CTDS storage managers. The real bonus is that this storage manager makes parallel IO possible for the bulk data.

#### 5.5. MonetDB

MonetDB (Boncz et al., 2009). is a distributed database system, developed at the CWI in Amsterdam, The Netherlands. It is the first database system using column stores and is aimed at fast data retrieval. Recently they developed SciQL (Zhang et al., 2012) for array based queries. It is a versatile language, more expressive than TaQL. MonetDB is not suited for holding interferometric data or image data because it is not designed to load such amounts of data. It is used very successfully to store the sources found in the imaging pipelines of LOFAR and to do source matching.

#### 6. Conclusions, lessons, and future developments

Over the last decade CTDS has proven itself at various radioastronomical institutes as a flexible and reliable data storage system. It has been used for various dataset types like visibility data, image data, calibration tables, and log tables. Some datasets have a size of only a few KBytes, others tens or hundreds of GBytes. The flexible data selection and access are powerful. TaQL gets positive feedback from users for its data selection, inspection, and manipulation functionality, although some people find the TaQL syntax a bit complex (as well as they find SQL complex).

The flexibility of the data managers proved to be very valuable. Both LOFAR and ALMA had a lot of benefit from the ability of using external data files. Furthermore, it made it easy to develop new experimental storage managers like the one based on ADIOS.

Standard CTDS uses Posix IO, but has the option of using memory-mapped IO. A nice lesson was that memory-mapped IO works well, but causes tools like top to report high memory usage when memory-mapping a large table. It leads to questions why the program uses so much memory.

Some lessons could be learned from using CTDS on a large global file system like Lustre (Schwan, 2003).

- CTDS supports concurrent table access (multiple readers, one writer). Lustre supports file locking, but performance can be degraded severely. Therefore the ability was added to bypass table locking.
- Many (small) files does not work well on Lustre. The very recently added MultiFile functionality solves this issue.

The Square Kilometre Array (SKA) telescope is the next generation radio telescope capable of producing TBytes of data per second. The analysis and initial design of its Scientific Data Processing (SDP) software has started in 2014. No decisions about data format and data storage have been taken yet. The MeasurementSet serves as a starting point for the interferometric visibility data model. The data storage system can be anything ranging from an Objectstore like Ceph,<sup>17</sup> the ADIOS system to CTDS. It will be interesting to see what the outcome is.

#### References

- Boncz, P.A., Manegold, S., Kersten, M.L., 2009. Database architecture evolution: Mammals flourished long before dinosaurs became extinct. PVLDB 2, 1648-1653, URL: http://www.vldb.org/pvldb/2/vldb09-10years.pdf.
- Codd, E.F., 1970. A relational model of data for large shared data banks. Commun. ACM 13, #6. de Vos, M., Gunst, A.W., Nijboer, R., 2009. The lofar telescope: System architecture
- and signal processing. In: Proceedings of the IEEE Special Issue Advances in Radio Telescopes
- Frings, W., Wolf, F., Petkov, V., 2009. Scalable massively parallel i/o to tasklocal file. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. URL: http://www.fz-juelich.de/ias/jsc/EN/ Expertise/Support/Software/SIONlib/node.html.
- Greisen, E.W., 2008. The aips interferometry data interchange convention. In: AIPS Memo Series, vol. 114. Greisen, E.W., 2012. Aips fits file format (revised). In: AIPS Memo Series, vol. 117.

Kemball, A.J., Wieringa, M.H., 2000. Measurementset definition v2.0. Casacore notes

- #229. URL: http://www.astron.nl/casacore/trunk/casacore/doc/notes/229.html. Liu, Q., Logan, J., Tian, Y., Abbasi, H., Podhorszki, N., Choi, J.Y., Klasky, S., Tchoua,
- R., Lofstead, J., Oldfield, R., Parashar, M., Samatova, N., Schwan, K., Shoshani, A., Wolf, M., Wu, K., Yu, W., 2014. Hello adios: the challenges and lessons of developing leadership class i/o frameworks. Concurrency Comput. Pract. Exp. 26, 1453–1473. doi:10.1002/cpe.3125. Pence, W.D., Chiappetti, L., Page, C.G., Shaw, R.A., Stobie, E., 2010. Definition of the
- flexible image transport system (fits), version 3.0. Astron. Astrophys. Pence, W., Seaman, R., White, R.L., 2012. A Tiled-Table Convention for Compressing
- FITS Binary Tables. ArXiv e-prints arXiv:1201.1340. Price, D.C., Barsdell, B.R., Greenhill, L.J., 2014. Is HDF5 a Good Format to
- Replace UVFITS? in: ADASS XXIV Proceedings, in: ASP series. ArXiv e-prints arXiv:1411.0507, in press.
- Schoenmakers, A.P., Renting, G.A., 2012. Measurementset description for lofar version 2.08.00. LOFAR wiki. URL: http://www.lofar.org/wiki/lib/exe/fetch. php?media=public:documents:ms2\_escription\_for\_lofar\_2.08.00.pdf. Schwan, P., 2003. Lustre: Building a file system for 1000-node clusters. In:
- Proceedings of the 2003 Linux Symposium.
- van Diepen, G.N.J., 2010. Table query language. Casacore notes #199. URL: http://www.astron.nl/casacore/trunk/casacore/doc/notes/199.html. van Diepen, G.N.J., Farris, A., 1993. The aips++ table data system", aips++ project. Viallefond, F., 2006. Alma science data model. In: ADASS XV. In: ASP Conf. Ser.,
- vol. 351. p. 627.
- Wells, D.C., Greisen, E.W., Harten, R.H., 1981. Fits: A flexible image transport system. Astron. Astrophys. Suppl. Ser. 44, 363–370. White, R.L., Greenfield, P., Pence, W., Tody, D., Seaman, R., 2012. Tiled Image
- Convention for Storing Compressed Images in FITS Binary Tables. ArXiv e-prints arXiv:1201 1336
- Zhang, Y., Scheers, L.H.A., Kersten, M.L., Ivanova, M., Nes, N., 2012. Astronomical data processing using sciql, an sql based query language for array data. ADASS XXI.

17 http://ceph.com.

Please cite this article in press as: van Diepen, G.N.J., Casacore Table Data System and its use in the MeasurementSet. Astronomy and Computing (2015), http://dx.doi.org/10.1016/j.ascom.2015.06.002

<sup>14</sup> http://www.paradigm4.com.

<sup>&</sup>lt;sup>15</sup> http://www.xldb.org/arrayql.

<sup>16</sup> https://github.com/SKA-ScienceDataProcessor/AdiosStMan.