Towards Scalable Ad-Hoc Climate Anomalies Search

CONFERENCE PAPER · NOVEMBER 2012
DOI: 10.1145/2447481.2447493

2 AUTHORS:

Peter Baumann
Jacobs University
115 PUBLICATIONS 526 CITATIONS

Dimitar Misev
Jacobs University
18 PUBLICATIONS 19 CITATIONS

Available from: Peter Baumann
Retrieved on: 23 June 2015
Towards Scalable Ad-Hoc Climate Anomalies Search

Peter Baumann
rasdaman GmbH
Hans-Hermann-Sieling-Str. 17
28759 Bremen, Germany
baumann@rasdaman.com

Dimitar Misev
rasdaman GmbH
Hans-Hermann-Sieling-Str. 17
28759 Bremen, Germany
misev@rasdaman.com

ABSTRACT
Meteorological data contribute significantly to “Big Data”; however, not only is their volume ranging into Petabyte sizes for single objects a challenge, but also the number of dimensions – such general 4-D spatio-temporal data cannot be handled through traditional GIS methods and tools. Actually, climate data tend to transcend these dimensions and add an extra time dimension for the simulation run time, ending up with 5-D data cubes.

Traditional databases, known for their flexibility and scalability, have proven inadequate due to their lack of support for multi-dimensional rasters. Consequently, file-based implementations are being used for serving such data to the community, rather than databases. This is recently overcome by Array Databases which provide storage and query support for this information category of multi-dimensional rasters, thereby unleashing the scalability and flexibility advantages for climate data management.

In this contribution, we present a case study where non-trivial analytics functionality on n-D climate data cubes has been established. Storage optimization techniques novel to standard databases allow to tune the system for interactive response in many cases. We briefly introduce the rasdaman database system used, present the database schema and practically important queries use case, and report preliminary performance observations. To the best of our knowledge, this is the first non-academic, real-life deployment of an array database for up to 5-D data sets.

Categories and Subject Descriptors
H.3.5 [Information Storage and Retrieval]: Online Information Services—Web-based services; J.2 [Computer Applications]: Physical Sciences and Engineering—Astronomy, Earth and atmospheric sciences

Keywords
Array databases, rasdaman, weather data, climate data

1. INTRODUCTION
Gridded meteorological data, such as forecasts and climate maps, consist of raster data describing various atmospheric parameters, such as air temperature, wind speed, and humidity. Dimensions of such a raster cube range from 2D x/y maps over 3D x/y/z and x/y/h cubes up to 4D x/y/z/t spatio-temporal data cubes, and even transcend this with 5-D cubes having two time axes (simulated time and simulation time). In terms of size, meteorological data certainly constitute “Big Data”, ranging into tens of Terabytes to multi-Petabytes already for a single cube.

Supercomputers running model simulations generate output describing temperature, humidity, wind speed, etc. in 3-D atmospheric space and over time. Although such data account for massive volumes, databases are not used for them, the main reason being that standard databases do not support the data structures on hand, namely multi-dimensional arrays. Consequently, the management of gridded meteorological data traditionally relies on ad-hoc file-based storage systems. Array DBMSs [11, 3, 17, 16, 20, 30] have set out to close this gap by providing a conceptual model and declarative query language on arrays, thereby enabling ad-hoc retrieval on “Big” sensor, image, and statistics data. There are manifold practical applications for Array DBMSs in Earth, Space, and Life sciences, but also in Social science as well as business and industry.

Today we observe how this novel technology gradually is getting adopted by scientific data and service providers. For example, the German meteorological office, Deutscher Wetterdienst (DWD) [28], provides forecasts for all of Germany on a routine basis, consisting of a variety of products for business and society at large. For the future, DWD is interested in tools allowing a wide range of not only preconceived, but also user-configurable products for increasing their service portfolio while decreasing the effort of establishing and maintaining it. To this end, DWD has contracted establishing a climate monitoring service based on an advanced Array DBMS, rasdaman [3, 15, 5, 1, 29]. Goal was to allow instantaneous analysis of the weather history for temperature and rainfall extremes, and in particular inspect incoming new weather time slices against the whole history; this was required to be possible for any given location. In this paper, we report about the rasdaman climate monitoring service resulting from this project.
The remainder of this paper is organized as follows. In the next section, we review related work, and then briefly introduce the rasdaman array DBMS in Section 3. Section 4 introduces the data sets used, while Section 5 presents storage organization and optimizations applied. In Section 6 the actual use case is presented by showing the database queries established; performance observations are listed in Section 7. Section 8 concludes the paper.

2. STATE OF THE ART

Typically, today’s gridded meteorological data, like forecasts or climate maps, are stored in files using suitable formats, such as Grib [32], NetCDF [22] or ESRI ArcGrid. Access is offered through proprietary interfaces or file-based tools like THREDDS (Thematic Realtime Environmental Distributed Data Services) [31]. THREDDS is “middleware aiming at simplifying the discovery and use of scientific data and to allow scientific publications and educational materials to reference scientific data. As such, it offers functionality to publish, contribute, find, and interact with data relating to the Earth system in a convenient, effective, and integrated fashion” [31]. While such systems allow for convenient metadata search (typically employing a relational database), files can only be downloaded, at best with some spatio-temporal subsetting from single files. No user-defined, ad-hoc processing or filtering is available at the raster level.

Ad-hoc filtering and processing is a strength of databases, due to their query language interfaces. Relational databases, though, currently offer no suitable modeling construct for raster data. BLOBs, as semantic-less bit strings, do not allow adequate operations, nor efficient storage (such as tiling). Array databases aim at closing this gap. While research has started already early on [3], only recently has this area emerged as a topic of common interest among database researchers. PostGIS Raster is an object-relational extension to PostgreSQL supporting 2-D arrays with array queries implemented as a “map algebra function” [20]. Such arrays constitute an additional attribute type. There is no built-in transparent tiling, queries have to consider tiling explicitly. Multi-dimensional arrays of unlimited size and with transparent tiling are characteristic for rasdaman. Implemented since 1995 [3] and with operational services hosting raster objects of two-digit Terabyte size it is considered the “most comprehensively implemented” system in Machlin’s review [17]. Recently, the EarthServer project [29] is establishing 100+ TB rasdaman databases with distributed peer-to-peer query processing.

SciQL is an array query language under development which extends the MonetDB column store [18]. SciDB is another recent approach aiming at an Array DBMS implementation [30], with an array model announced similar to SciQL. SciQL and SciDB differ from other Array DBMSs in that arrays are not modeled as a new attribute type, but are treated similar to tables. In practice, this entails some problems like unstable schemas and the inability to use standard referencing means (i.e., foreign keys) for relating data and metadata.

The aforementioned approaches only allow a regular array partitioning, called “chunking”; rasdaman, on the other hand, supports arbitrary partitioning into sub-arrays, controlled by a storage layout language [5] which also serves to handle further physical parameters.

Recently, activities have started to extend the ISO SQL:2003 standard [13] with advanced array analytics support. Currently, SQL supports only 1-D arrays with basically two operations only, subsetting and merging. The rasdaman query language is a candidate proposed to ISO.

In the domain of geo-services, there is already such a standard, the Open Geospatial Consortium (OGC) Web Coverage Processing Service (WCPS) [3, 4]. This standard defines a multi-dimensional raster processing language embedded in the OGC suite of modular standards; raster data there form a part of the more comprehensive data structure of coverages [14]. The main functional extension of WCPS over rasdaman is the ability to handle geo-referenced objects and coordinate reference systems. Syntactically, WCPS is more aligned with XQuery, with the rationale that geo-metadata commonly tend to move towards XML representations [10]. In the EarthServer initiative [29], WCPS is being coupled with W3C XQuery [9] to enable unified querying of coverage and metadata. The syntactic preparation of WCPS is expected to ease this transition.

3. THE RASDAMAN ARRAY DBMS

We give a brief introduction to the rasdaman data and query model; for details we refer to the language guide [21], scientific publications [3, 5], and the project’s web site [1]. The query language, rasql, allows composing expressions on arrays embedded into the select / from / where flavor of SQL.

3.1 Conceptual Model

In the data model, arrays appear as a special attribute type, parametrized with dimensional extent and the cell (“pixel”, “voxel”) type. The extent of a d-dimensional array, its spatial domain, is given by the cross product of the intervals given by lower and upper bounds $lo_i$ and $hi_i$, for each axis $0 \leq i < d$. Boundaries span the integer domain, allowing negative indices as well; this is useful, for example, when modeling $n \times n$ filter kernels which represent matrices spanning from $-n$ to $+n$. Also, extensible maps may get enlarged on any side, easily leading to negative indices when extended in West or South directions. Within this domain, each integer-valued vector addresses an array cell holding some atomic or composite value.

A rasdaman table (called a collection, following the ODMG standard [6]) has two columns holding a system-maintained OID and the array itself. This allows for foreign key references as usual, something which typically happens, e.g., in satellite image archives where millions of satellite images are stored, each with an associated metadata record (Figure 1).

First, a cell type has to be chosen. All common types known from programming languages are supported, plus single and double precision complex numbers. Additionally, nested structs can be defined at system run-time.

For example, a weather data set containing rainfall, temperature, and wind speed values might rely on cell structures like this:

\[ \text{Chosen randomly for the purpose of this example} \]
typedef struct {
    double rainfall;
    double temperature;
    double windspeedX, windspeedY;
} WeatherData;

Next, an array type is defined. Actually, array types constitute second-order elements (like templates in C++) which are parametrized with cell type and spatial extent. For each boundary individually, either a fixed bound or a wildcard (denoted by an asterisk, "*" ) can be indicated, thereby controlling query-time boundary checking and thus enabling dynamically growing arrays.

Consider as an example for an array type the output of a numerical weather prediction model. Such a model produces weather forecasts for a region at several vertical levels and a number of forecast times. One model that produces forecasts for Europe is called COSMO-EU (Consortium for Small Scale Modelling) [27]. The COSMO-EU model is DWD’s most important numerical model for short range weather prediction because it determines forecasts for up to 78 hours. The grid of this model has a mesh width of 7 km which results in an average size of 49 square kilometres for each grid cell. The grid consists of 665×657 = 436,905 points. The model is calculated on 40 vertical levels with the lowest level 10 meters above ground and the uppermost level in a height of 24 km. The following definition will fit this situation. The axis sequence is x/y/z/t:

typedef marray<
    WeatherData, [ 0:664, 0:656, 0:39, 0:77 ]
> WeatherCube;

Finally, a set type is constructed over this array type, which can be instantiated to obtain collections of such arrays:

typedef set< WeatherCube > WeatherSet;

This three-step approach is inherited from the historical decision to align rasdaman with the ODMG standard which, in turn, orients itself on compilation-based programming techniques; for the future, a more SQL-style approach is foreseen in which the above example might be expressed as:

```
create table WeatherObservations(
    id: integer not null,
    creationTime: date,
    weatherCube: array(
        rainfall: number,
        temperature: number,
        windspeedX: number,
        windspeedY: number
    ) [ 0:664, 0:656, 0:39, 0:77 ]
);
```

An important feature needed for the DWD use case is handling of NULL values. As opposed to the ISO SQL standard [13], which foresees one null value, scientific communities have to deal with potentially several null values representing measurement situations like “bad reading”, “value out of range”, “no value delivered”, “sensor unavailable”, etc. This provides a platform to support in the future, e.g., the OGC Sensor Web Enablement (SWE) standards like the Observation and Measurement specification [7].

To capture this, the rasdaman type definition allows defining a set of NULL values which will be respected when executing operations on the corresponding array instances. For instance, assuming values 990, 997, 998, and 999 for the null situations above, this might be indicated in the collection type through a list of values and – to abbreviate longer lists – as intervals as follows:

typedef set< WeatherCube >
    null values [ 990, 997:999 ]
WeatherSet;

### 3.2 Query Language

We introduce the query language [21] with examples from the climate data domain. A ‘—’ prefix starts comments in a query.

*Trimming* extracts a subset of the same dimension as the input array A. For each dimension, lower and upper bound of the cutout are indicated, e.g.:

```
A[ x0:x1, y0:y1 ]
```

*Slicing* also delivers a cutout, however with a reduced dimension. For example,

```
A[ x0:x1, y0 ]
```

delivers a 1-D array with closed interval (x0,x1) as its domain, obtained by slicing A at position y0. Slicing d-dimensional array d times delivers a single array cell, by definition 0-D
Induced operations apply some unary or binary operation, identical domains, \(A+B\) simultaneously. For example, for two arrays \(A\) and \(B\) with identical domains, \(A+B\) delivers a cell-wise addition of \(A\) and \(B\). Induced by the cell type’s operations, the query language offers all these boolean, arithmetic, logarithmic, and trigonometric operations also on the array level. Null values in both operands are respected in a straightforward manner; the result object has the union of both null value sets, and null results in cells take on one such result null value in a non-deterministic way.

The \textit{condense} operation resembles array aggregation, similar to the relational aggregates\(^2\). For example,

\[
\text{max_cells}(A)
\]

calculates the maximum of all cell values in \(A\). Condensers allow to derive min, max, average, etc.; likewise, existential and universal quantifiers are available through \textit{some_cells}(A) and \textit{all_cells}(A). Null values are ignored in the aggregation process.

The query language overall is based on three core algebra operations. A general \textit{marray} allows to construct and initialize an array; a declarative array iterator inspects all elements in the array domain and allows setting the cell value dependent on the iterator position. For example, we can define an induced addition through

\[
A + B := \text{marray x in sdom(A)} \\
\text{values A}[x] + B[x]
\]

The general \textit{condense} operation aggregates over cells in an array using an array iterator. For example,

\[
\text{condense +} \\
\text{over x in sdom(A)} \\
\text{using A[x]}
\]

calculates the sum of all cell values in \(A\).

As a final example, the histogram of an \(n\)-D array \(A\) with 8-bit integer pixels can be written as

\[
\text{marray bucket in sdom(A)} \\
\text{values count_cells( A = bucket )}
\]

In rasql, such operations on array-valued attributes are embedded in SQL-style set-oriented queries. Consider the following (hypothetical) example, which also illustrates result encoding and the use of array predicates in a where clause: “for all simulations conducted where temperature difference exceeds 10\(^\circ\) somewhere, deliver windspeed at time \(T\), encoded in NetCDF”. The corresponding query is:

\[
\text{select netcdf(} \\
\text{sqrt( sq( w.windspeedX )} \\
+ sq( w.windspeedY ) \\
\text{)} \\
\text{) from WeatherSet AS w} \\
\text{where max_cells( w.temperature )} \\
- \text{min_cells( w.temperature )} \\
> 10
\]

Both select, insert, update, and delete operations are available in rasql. A specific extension to the update statement is the partial update operation which allows updating arrays piecewise. Altogether, this achieves a smooth embedding of array handling in a standard relational system.

3.3 Architecture

Physically (and transparent on the query level), arrays are partitioned into contiguous, non-overlapping sub-arrays called \textit{tiles}. Tiles are stored in database BLOBs and serve as units of access during query evaluation. Multi-dimensional indexes help to quickly determine the tiles affected by a query window; among the methods used is the R+ tree [25]. This allows for not only regular grids, but an arbitrary partitioning into (currently non-overlapping) tiles, thereby adjusting better to arbitrary query windows. Adjusting tiling to query access patterns constitutes an important tuning mechanism.

The tiled storage architecture suggests an evaluation strategy of tile streaming to minimize main memory needs. Further, this paradigm lends itself well towards tile-parallel query evaluation.

For persistent storage, rasdaman utilizes a pre-existing relational DBMS. This allows using all relational facilities for metadata management in conjunction with the array capabilities.

The relational schema of the rasdaman data consists of a BLOB table holding the array tiles, and several further tables holding rasdaman catalog information, array indexes, etc. A simple driver interface connects the rasdaman server to the underlying DBMS, which makes rasdaman insensitive to the DBMS used. Several open-source and commercial systems, as well as pure file-system storage, are currently supported.

4. CLIMATE EXTREMES DETECTION

In a project initiated by the German meteorological office, Deutscher Wetterdienst (DWD) [28], rasdaman has been chosen for implementing a service for statistical evaluations on climate data time series with the goal of detecting and assessing climate anomalies. Concretely, based on the rasdaman array database management system, \(p\)-percentiles have to be derived from the base data for a parameter \(p\) submitted as part of an ad-hoc query. Further query input parameters are (i) the area over which the percentiles are to be determined, given by a bounding box, and (ii) the time periods to be investigated, such as months, seasons, or arbitrary monthly intervals.

Data used consist of 3-D and 4-D timeseries cubes; for the
The topic of this paper the following data sets are relevant:

- the E-OBS dataset [19] as provided by KNMI (Fig. 2); and
- the GPCC dataset [23, 12, 8] provided by the Global Precipitation Climatology Centre (Fig. 3).

This has been condensed to form suitable time series input data for percentile computation. Further, resolution has been homogenized by down-sampling some data available in 0.5° data to the common and requested resolution of 1°. While this is not essential for rasdaman from a technical viewpoint (as multi-dimensional scaling is supported), it was a requirement of DWD.

For data ingest, the OGC WCS-T standard (Web Coverage Service - Transactional) [26] ideally could have been used. However, as an up-to-date WCS-T specification currently is not available from OGC (revamping it will be addressed by the EarthServer project), ingest had to resort to custom-made shell scripts. These scripts insert metadata (where provided) through SQL statements and import the raster data through rasql statements.

A single object in the file system is made up from several files. During ingest, these files are combined into a single raster object in the database using a so-called “partial update” query in rasdaman. The following – simplified – command exemplifies addition of a time slice to a data cube:

```
rasql -q "update $c as m
  set m[ $time, 0:$xhi, 0:$yhi ]
  assign (short) inv_netcdf($1)"
--file myFile.nc
```

Note that the (shell-escaped) “$1” stands for an input parameter which is transported to the server alongside with the query - in this case, the NetCDF file indicated in the -file parameter. Prior to storage in the database, the cell type of the NetCDF input is converted to a short integer value using the case operator.

The E-OBS dataset consists of four components (“variables”) – daily mean, minimum, and maximum temperature, and daily precipitation – covering the area of Europe (40°W - 75°E × 25°N - 75°N) from 1951 until today. Data are available on a 0.25 and 0.5 degree regular lat-lon grid, as well as on a 0.22 and 0.44 degree rotated pole grid. For our use case the 0.5 degree regular lat-long grid data has been used, which translates to 22,644 measurements of horizontal extent 101 × 232. As explained in Section 5.1, this has been remodeled from 3-D to a 5-D structure to allow easier handling of time queries.

Precipitation has an important role in the global climate, and analysis of long-term precipitation data allows to assess climate change and its impact on all spatial scales. To this end the Global Precipitation Climatology Centre (GPCC) has been established by Deutscher Wetterdienst as a German contribution to the World Climate Research Programme (WCRP). Based on in situ raingauge data, the GPCC provides gridded monthly precipitation data sets covering the Earth’s land surface in various resolutions [24]. We have used GPCC precipitation data from 1951 to 2010 in 0.5° × 0.5° resolution.

**Figure 2:** Visualization of E-OBS mean temperature data covering one month of measurements

**Figure 3:** Visualization of GPCC precipitation data covering measurements over 1951

### 5. TUNING THE USE CASE

Several factors obviously determine overall performance, including (but not limited to) data load times from disk; query processing speed; main memory utilization (i.e., avoiding swapping); and client-server transfer times. Fortunately, in this use case both data and query workload are well known, so database tuning can focus on several aspects. We first discuss a “time axis explosion” for a better handling of calendar units, then present a data partitioning scheme tuned towards the query workload given, and finally outline means for speeding up processing in main memory. As the axis explosion approach modifies queries, for didactical reasons we first present this concept before detailing the actual queries in Section 6.

#### 5.1 Calendar Handling

As calendar time is highly non-regular – different lengths of months, leap years, etc., a specific modeling was chosen to simplify queries. It consists of extra time dimensions
to obtain one dimension per granularity. In such a setup, a cube has dimensions year, month, day, followed by the spatial dimensions x and y.

Several queries address calendar time. A problem with time axis is that aggregated time units, such as months and years, are highly non-regular. For rasdaman this means that regular subsetting (as in the examples shown above) and scaling do not work. In the future, this will have to be addressed as a specific research issue; for now, a specific modeling approach was chosen to simplify queries and avoid irregular spacings. To this end, the single time axis has been unfolded into further time axes, each one representing one level of granularity of extra time dimensions to obtain one dimension per temporal granularity unit. In such a “ragged” setup as shown on Figure 4, a cube has the dimensions year, month, day, followed by the spatial dimensions x and y. Cells going beyond the day limit in a month are set to null. This allows convenient access on all levels of detail by simply using the subsetting and aggregating capabilities of the rasdaman query language.

5.2 Storage Layout
Section 3.3 introduced the concept of tiles in rasdaman, and now we look at how this was applied to the particular use case at hand. The rasdaman query language comes with a storage layout extension that allows precise control of how arrays are partitioned and stored as tiles at ingestion time [5]. This sub-language has support for regular tiling where all tiles are of same size, directional tiling that optimizes access along dimensions of interest, tiling based on areas of interest, statistical tiling that automatically adapts the tiling structure based on access pattern, etc.

Since this use case involves a time series, the most optimal tiling strategy is a restricted version of directional tiling along the time axis, which allows fast access of single pixels or an area through time. It is restricted in the sense that tiles do not span the time axis from start to end, as retrieving the data at a single time point in such case (which could certainly come up in a query) would require fetching all tiles from the database. Instead, tiles span 50 points along the time axis, which offers a good compromise between both access patterns. Figure 5 illustrates this in a simplified way, comparing typical tiling strategy (e.g. files for each time step on the filesystem), with the optimized tiling in rasdaman.

5.3 CPU Optimization
Array queries are typically CPU-bound, so intelligently utilizing the available CPU resources has great impact on query response times. Where applicable, intra-query parallelization is used to ensure that multi-core processors are all involved, resulting in a largely improved performance compared to a sequential evaluation. Translating query fragments to native executable code by just-in-time compilation further cuts evaluation time [15].

6. CHANGE DETECTION QUERIES
The goal in this use case has been to detect possible temperature / precipitation anomalies over a certain period of time, taking into account data from the past 60 years. This involves several steps of computation to produce a map that can easily show the anomalies:

1. Compute aggregate values (average, minimum, maximum) for each pixel separately over the specified time period.
2. Compute the percentile value, e.g., the 95th percentile, for each pixel over the whole data since 1951.
3. Mark the pixel’s where the aggregated values are above or below the accordingly computed percentile value.

The following examples illustrate the results obtained with several different parameters.

Q1 “Compute the first tertile, for the current year.”

```
select quantile( c[ sdmon(c) sdom(c) [0].hi, -- year
  **:* -- month
  **:* -- day
  **:* -- lon
  **:* -- lat
  ],
3 ) -- get tertiles
[0] -- first component of result array
from eobs as c
```
This is a simple query that illustrates how the quantile function in rasdaman can be used for computing $q$-quantiles. It accepts an array expression and the $q$ value, returning back an array of values for budget 0 to $q - 1$, from which values can be extracted via slicing/subsetting, just as from any regular array. Auxiliary function sdom() returns the spatial domain of the array, so we can extract the upper limit that corresponds to the current year.

Q2 “Show where temperatures during the last year have been below the first tertile since 1951”. This query results in the map shown on Figure 6.

Figure 6: Percentile map showing areas with temperatures below the first tertile for the current year.

This map is generated with a more complex query that can be broken down to the sub-queries below.

1. Compute aggregate values for the current year (let us call this array $A$):

   ```
   select
   marray i in [ 0 : sdom(c)[3].hi, 0 : sdom(c)[4].hi ]
   values
   avg_cells( c[ sdom(c)[0].hi, *:*:*:*:*:*:*:*:*:* ] )
   from eobs as c
   ```

2. Compute the first 3-quantile over the previous years (array $Q$):

   ```
   select
   marray i in [ 0 : sdom(c)[3].hi , 0 : sdom(c)[4].hi ]
   values
   quantile( c[ *:*:*:*:*:*:*:*:*:* ]
   *:*, *:*:, i[0], i[1] ] )
   from eobs as c
   ```

3. Mark the areas where the aggregated values are below the first tertile:

   ```
   select
   png( A * {1c, 1c, 1c} )
   overlay
   (A < Q) * {255c, 0c, 0c}
   from A, Q
   ```

Q3 “Where did the temperatures during the last year for the interval February through June exceed the 90th percentile, since 1951?” The result is shown in Fig. 7, and shows the areas where the weather has been warmer during these months last year than in the past 60 years.

Figure 7: Percentile map showing “hot” areas.

Q4 “Colored classification map for some given quantile $q$.” Concretely, $q = 100$ (percentiles) where values below the 10th percentile appear in blue and above the 90th are marked in red. Figure 8 shows the query result.

```python
select
-- convert to greyscale
( (char) ( (c[0,1,9,*:*:*:*] + 9999) / 55 ) *
{1c,1c,1c}
)
overlay
-- mark with red above the 90th percentile
( (c[0,1,9,*:*:*:*] >
quantile(c[0,1,9,*:*:*:*], 100) [89]
) *
{255c,0c,0c}
)
overlay
-- mark data below 10th percentile in blue
( ( c[0,1,9,*:*:*:*] < quantile(c[0,1,9,*:*:*:*], 100) [9]
and
 c[0,1,9,*:*:*:*] > -9999
) *
{0c,0c,255c}
)
from eobs as c
```

Figure 8: Colored classification map for some given quantile $q$
Where did precipitation in July and August 2010 exceed the 99th percentile, since 1951?” The query for this example is essentially same as Q2, except that it is now applied to the GPCC data set, and cut-out of the whole map focusing on South Asia has been selected (Figure 9).

It can be noticed that the area of North Pakistan has been marked as having extremely high precipitation when analyzed historically, which caused the worst flood in the history of Pakistan in late July and August.

7. PERFORMANCE OBSERVATIONS

The system installation provided by DWD consists of a standard PC with a 4-core Intel Xeon X5680 processor with 4GB RAM running SUSE Linux Enterprise 11. On this machine, PostgreSQL and rasdaman were installed as database servers, together with standard components like an Apache Web server. Measurements were conducted via command line so that network transmission times can be neglected.

<table>
<thead>
<tr>
<th>Query</th>
<th>Runtime in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0.98</td>
</tr>
<tr>
<td>Q2</td>
<td>51.26</td>
</tr>
<tr>
<td>Q3</td>
<td>38.16</td>
</tr>
<tr>
<td>Q4</td>
<td>0.13</td>
</tr>
<tr>
<td>Q5</td>
<td>21.25</td>
</tr>
</tbody>
</table>

Table 1: Runtime of sample queries.

Note that Q1 and Q4 determine climate change indicators for data covering a single year; with a response time below one second, they allow interactive requests. Calculating percentiles over the whole data takes below one minute for Q2, Q3, and Q5; these represent monthly updates of the whole data over Europe and the World, so they can be provided by DWD now shortly after arrival of the new data.

Due to the limited contract, there were no resources available for a thorough performance evaluation— it was sufficient to deliver a solution with satisfying performance. In future, we plan to catch up on this by involving the Large-Scale Scientific Information Systems research group at Jacobs University, in the course of a research collaboration.

8. CONCLUSION

The use cases undertaken have shown convincingly to DWD that array databases like rasdaman indeed provide an added value through the flexibility of the query language combined with the tuning capabilities of the tiled storage organization, achieving response times at least competitive with current file-based evaluation. In many of the use cases chosen this effectively enables real-time retrieval.

Hence, the contribution of this paper to the geo services community is to demonstrate (i) that multi-dimensional database support establishes a distinct convenience by treating all spatial and temporal dimensions alike and (ii) how tiling and the concept of “time axis explosion” can achieve interactive speeds for non-trivial queries on multi-dimensional objects. In particular, to the best of our knowledge this is the first non-academic use of array database technology, with datasets of up to five dimensions.

We conject, therefore, that array databases provide distinct advantages—such as flexibility, scalability, and information integration—over architectures where only metadata sit in databases and data are kept in flat files. In detail, we see several distinct advantages in using an array database like rasdaman:

1. Data alignment on disk can be adjusted to query patterns. This has a potential of improving access times substantially.
2. Versatile on-the-fly processing is possible, beyond mere subsetting.
3. Multi-dimensional data can be handled smoothly— as opposed to traditional GIS-based tools, an array database treats time just like any other (spatial or non-spatial) axis.
4. As the query language is “safe in evaluation” (every query is guaranteed to terminate after a finite number of steps) the query interface on principle can be opened to external users (although possibly combined with quota).
5. Unified service management is possible, achieving tight information integration of data and metadata in the same database. Improved consistency and easier administration are core benefits achieved.

It should be noted, though, that despite these encouraging results this is not a comprehensive benchmark: neither do the queries challenge the language systematically, nor do runtime measurements establish a rigorous benchmark. Rather, we see the value of this contribution as an immediately practice-driven case study based on the demands of DWD’s day-to-day business requirements of presenting a growing variety up-to-date meteorological products to their clients.

Also, the marray operator deserves more attention with respect to performance; due to its generality it is not easy to optimize as such, so this resembles a significant research challenge.
Further, we plan to address the following issues, in collaboration with our research partners:

1. Establishing Web front-ends to encapsulate database functionality into diversified clients for various user groups, ranging from end users (demanding simple, yet visually appealing point-and-click interfaces which hide the query language) to power users (which require access to the query language for establishing their custom-made queries).

2. Establishing best practices on time modeling by way of an in-depth investigation of the modeling alternatives and their pros and cons.

3. Establishing an operational installation for multi-dimensional climate data. As documented ingest procedures are available, an extension to larger datacubes is no problem on principle. However, automated ingest procedures, including detailed logging and alerting, should be established. Further relevant issues include database backup, access rights management, administration

Another extension requested by various data centers is to avoid copying the whole archive for retrieval. To this end, Alagiannis et al. have established the concept of in-situ support [2], i.e.: answering queries on data without prior import, but operating directly on the native file organization of the data center's archive. Such a feature is currently under implementation in rasdaman.

9. ACKNOWLEDGEMENT

This work has been partially supported by Deutscher Wetterdienst (DWD, www.dwd.de). The technical and scientific support provided by Dr. Jürgen Seib, Dr. Peter Bissolli, and Karsten Friedrich, all DWD, is gratefully acknowledged.

10. REFERENCES


