Testbed-12 Big Data Database Engineering Report

Table of Contents

Publication Date: 2017-06-30

Approval Date: 2017-06-29

Posted Date: 2016-04-07

Reference number of this document: OGC 16-036r1

Reference URL for this document: http://www.opengis.net/doc/PER/t12-A076

Category: Public Engineering Report

Editor: Christian Autermann

Title: Testbed-12 Big Data Database Engineering Report

OGC Engineering Report

COPYRIGHT

WARNING

This document is an OGC Public Engineering Report created as a deliverable of an initiative from the OGC Innovation Program (formerly OGC Interoperability Program). It is not an OGC standard and not an official position of the OGC membership.It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements. However, the discussions in this document could very well lead to the definition of an OGC Standard.

LICENSE AGREEMENT

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD. THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER’S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by

destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Abstract

The amount of (geospatial) data collected and transferred is rapidly increasing. The purpose of this ER is to describe options and recommendations for the delivery of large amounts of data as database delivery. This ER therefore describes and evaluates different aspects of this challenge:

- Data management: How to organize large amounts of data so that it can be efficiently accessed through OGC service interfaces?
- Encoding: Transferring large amounts of vector data in XML based formats (e.g. GML, O&M) leads to specific challenges as the parsing of large XML files is often problematic.

Available implementation: Several software packages exist to handle large amounts of geospatial data. We will investigate to which these approaches are in-line with OGC standards or how standards compliance could be achieved.

The evaluation and findings in the related Big Data Tile Database Implementation are documented in this ER as well. The objective of this ER is to provide recommendations of how the delivery of large amounts of raster data as database delivery can be considered within OGC specifications and future activities.

Business Value

The approach described in this ER provides owners of GeoPackage tile archives to cope with data volumnes for which SQLite-based GeoPackages reach their limit while simultaneously retaining the established GeoPackage data model.

What does this ER mean for the Working Group and OGC in general

Currently, a common approach based on existing OGC standards for the exchange of large tile data stores is missing. This ER presents a solution based on the OGC GeoPackage standard. On the one hand, this approach is thus an example how OGC standards may be applied to large datasets (Big Data DWG) and provides on the other hand a standards-based solution that is considered relevant for the whole OGC beyond the working group’s scope.

How does this ER relate to the work of the Working Group

This ER presents a novel approach for handling large tile datasets in Geopackages. Since the scope of the Big Data DWG is handling large datasets with existing OGC standards and beyond, the work presented in this ER is of high relevance for this group.

Keywords

ogcdoc, testbed-12, bigdata, geopackage, array database, raster, sqlite, tiles, storage, exchange, sensorweb

Proposed OGC Working Group for Review and Approval

This Engineering Report will be submitted to the Big Data Domain Working Group for review and comment.

Chapter 1. Introduction

1.1. Scope

This OGC® document defines an alternative application of the GeoPackage data model to another database management system, namely PostgreSQL, in order to facilitate the exchange of large tile data stores.

It further evaluates the performance and addresses possible caveats of this transition and discusses alternative approaches of handling large tile data sets.

1.2. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Table 1. Contacts

Name	Organization
Christian Autermann	52°North GmbH
Simon Jirka	52°North GmbH
Stephan Meissl	EOX IT Services GmbH
Peter Baumann	Jacobs University

1.3. Future Work

No future work is planned to this document.

1.4. Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Chapter 2. References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

- OGC 06-121r9, OGC® Web Services Common Standard

NOTE: OWS Common Standard contains a list of normative references that are also applicable to this Implementation Standard.

- OGC 12-128r11, OGC® GeoPackage Encoding Standard - With Corrigendum, http://www.geopackage.org
- OGC 08-068r2, OGC® Web Coverage Processing Service (WCPS) Language Standard
- OGC 09-110r4, OGC® WCS 2.0 Interface Standard - Core, version 2.0.1
- OGC 09-146r3, OGC® Coverage Implementation Schema
- ISO 9075 SQL Part 15: MDA (Multi-Dimensional Arrays)
- OGC 06-042, OpenGIS® Web Map Server Implementation Specification
- OGC 10-025r1, Observations and Measurements - XML Implementation
- OGC 10-004r3, OGC® Abstract Specification: Geographic information — Observations and measurements
- OGC 09-000, OGC® Sensor Planning Service Implementation Standard
- OGC 06-028r5, OGC® Sensor Alert Service Implementation Specification
- OGC 08-133, OpenGIS ® Sensor Event Service Interface Specification
- OGC 13-131r1, OGC® Publish/Subscribe Interface Standard 1.0 - Core
- OGC 14-065, OGC® WPS 2.0 Interface Standard Corrigendum 1
- OGC 12-000 OGC® SensorML: Model and XML Encoding Standard

Chapter 3. Terms and definitions

For the purposes of this report, the terms and definitions given in the above references apply. In addition, the following terms and definitions apply.

3.1. GeoPackage file

A platform-independent SQLite database file that contains GeoPackage data and metadata tables with specified definitions, integrity assertions, format limitations and content constraints.

3.2. tile

A rectangular pictorial representation of geographic data, often part of a set of such elements, covering a spatially contiguous extent and sharing similar information content and graphical styling, which can be uniquely defined by a pair of indices for the column and row along with an identifier for the tile matrix.

3.3. tile matrix

A collection of tiles for a fixed scale.

Chapter 4. Conventions

4.1. Abbreviated terms

ADA:WG
: Array Database Assessment Working Group

DBMS
: Database Management System

ECMWF
: European Centre for Medium-Range Weather Forecasts

ESA
: European Space Agency

GDAL
: Geospatial Data Abstraction Library

GPKG
: GeoPackage

GPKG-PG
: PostgreSQL-GeoPackage

GPU
: Graphics processing unit

ISO
: International Organization for Standardization

IEEE
: Institute of Electrical and Electronics Engineers

JPEG
: Joint Photographic Experts Group

LSA
: Large Scale Analytics

MDA
: Multi-Dimensional Arrays

NCI
: National Computational Infrastructure

NIST

National Institute of Standards and Technology

OGC
: Open Geospatial Consortium

O&M
: Observations and Measurements

PG
: PostgreSQL

PNG
: Portable Network Graphics

RDA
: Research Data Alliance

RPM
: RPM Package Manager

SAS
: Sensor Alert Service

SDI
: Spatial Data Infrastructure

SES
: Sensor Event Service

SOS
: Sensor Observation Service

SPS
: Sensor Planning Service

SQL
: Structured Query Language

SWE
: Sensor Web Enablement

WKB
: Well Known Binary

Chapter 5. Overview

A common approach for exchanging large tile data stores is missing at the moment. To address this issue, this ER defines an alternative application of the GeoPackage data model to another database management system, namely PostgreSQL, and evaluates the performance.

It also addresses possible caveats of this transition and discusses alternative approaches of handling large tile data sets.

The document is structured as follows: section 6 lists the functional requirements for an exchange of large tile data stores. Section 7 describes the use cases on global, regional and local/operational level. Section 8 describes the core contribution and introduces the approach for utilizing geopackages for the exchange of large tile datasets. It also presents the results of evaluation.

The two following subsections describe two alternative approaches for managing and handling large amounts of spatial data: section 9 introduces the concepts of array databases. Section 10 provides an overview on Sensor Web technologies and discusses challenges to be addressed for a handling of large amounts of data. Finally, section 11 lists recommendations to OGC based on the work described in this ER.

Chapter 6. Status Quo & New Requirements Statement

Since a common functionality for exchanging large tile data stores is currently missing, the required functions for the exchange of tile data stores are as follows:

- Definition of file format(s) (database format(s)) for storage and exchange of large (global/regional) tile stores.
- Storage of data to support retrieval and dissemination of large data to include large tiled data.
- Support of automated simplified access, retrieval, and dissemination of multiple tiles at a time (by a bounding box, zoom levels (e.g. WPS, WMTS, GPKG, shell scripts, etc.).
- Distribute large amount of tiled data in a non-proprietary format/database utilizing the same tiling scheme, CRS, etc. based upon the common profile of OGC standard.
- Provide recurring updates from global to regional databases.
- Provide on-demand updates of data and data tiles to regional and operational/local users.
- Support complex analytics across these heterogeneous global and regional big data collections.

Chapter 7. Use Cases

There are three tiers of users with different data requirements: global, regional, and operational/local.

Global oversight is responsible for maintaining and updating relevant geospatial information in multiple format (base imagery, raster digital maps, vector databases, coverages including elevation data, etc). The global big data archive is managed and updated as part of daily activities.

Regional planners are focused on supporting potential operations within their region. They require relatively current subset replicated and updated from the global database on a regular basis (e.g. every three months).

Operational and local users will conduct operations. Their geospatial requirements vary depending on the specific activity and area. When committed, they require all relevant geospatial data to be packaged and pushed to them quickly. Their mission profiles will often be defined by a large bounding box and tile requirements so they can receive and download information and then operate in a low-bandwidth or disconnected mode.

Chapter 8. Extracting GeoPackages from a large Database

This sections summarizes the evaluation and findings in the related Big Data Tile Database Implementation task. The objective is to provide recommendations how the distribution of large amounts of RGBA raster data as database delivery can be considered within OGC specifications and future activities.

In the Big Data Tile Database Implementation task PostgreSQL has been evaluated to serve as alternative to SQLite for a container of tiles as specified in the GeoPackage standard [OGC 12-128r11]. The evaluation considered storing binary data directly in PostgreSQL as well as using the raster functionality of PostGIS.

GeoPackage describes itself as: "an open, standards-based, platform-independent, portable, self-describing, compact format for transferring geospatial information". The GeoPackage standard describes a set of conventions for storing vector features, tile matrix sets of imagery and raster maps at various scales, and extensions within an SQLite database. This section deals with tile data only. Tiles are encoded in either JPEG or PNG depending on the requirement for transparency and/or lossless compression and are stored in SQLite directly as binary data.

Based on the evaluation SQL and Python shell scripts have been developed for PostgreSQL database generation, data loading, data dumping, and database content dropping. These scripts have been provided to the Testbed-12 LSA thread and tasks like A018 WPS server that provides tiles access as well as A042 WMTS server to serve image tiles could evaluate using them.

Finally those scripts are used for evaluating and comparing the solution to the original SQLite usage.

At the Testbed-12 kick-off the motivation to look at PostgreSQL has been discussed. One of the questions was, if it might not be better to evaluate the splitting of large SQLite files into multiple ones instead. While this is considered to be a viable alternative an evaluations is out of scope of the activities performed in the frame of Testbed-12.

8.1. Implementation Overview & Usage Quickstart

The software package is named PostgreSQL-GeoPackage and available online at http://github.com/EOX-A/PostgreSQL-GeoPackage [14].

The software package holds scripts used to evaluate the suitability of PostgreSQL to serve as alternative to SQLite for a container of raster tiles as specified in the GeoPackage standard [OGC12-128r11].

Presumably the quickest and easiest way to evaluate the implementation is to use the provided Vagrant configuration. Just follow the Vagrant instructions [15] for a clean environment and connect to it:

```
$ vagrant ssh
$ cd PostgreSQL-GeoPackage/
```

Create a PostgreSQL database and load the GeoPackage schema into the PostgreSQL-GeoPackage:

```
# create the gpkg role
$ createuser --createdb --superuser gpkg

# create the gpkg database
$ createdb --encoding UTF8 --username gpkg gpkg

# import the GeoPackage schema
$ psql --username gpkg --dbname gpkg --file gpkg-pg_init.sql
```

Load a SQLite GeoPackage into the PostgreSQL-GeoPackage, dump it again, and validate the result of the round-trip:

```
$ NAME=Sample-GeoPackage_Sentinel-2_Vienna_Austria

# load the SQLite GeoPackage
$ ./gpkg-pg_loadpkg.py ${NAME}.gpkg "dbname='gpkg' user='gpkg'"

# dump the SQLite Geopackage to plain text
$ sqlite3 ${NAME}.gpkg .dump > before.sql
$ rm ${NAME}.gpkg

# dump the PostgreSQL GeoPackage to a SQLite GeoPackage
$ ./gpkg-pg_dump.py "dbname='gpkg' user='gpkg'" ${NAME}

# dump the exported SQLite Geopackage to plain text
$ sqlite3 ${NAME}.gpkg .dump > after.sql

# show the difference between the original and exported files
$ diff before.sql after.sql
```

Dump a spatial subset of the PostgreSQL-GeoPackage and validate it by visual comparison to a GDAL generated subset:

```
$ NAME=Sample-GeoPackage_Sentinel-2_Vienna_Austria

# create a subset with gdal
$ gdal_translate -of GPKG -srcwin 768 768 256 256 ${NAME}.gpkg ${NAME
}_subset_gdal.gpkg
$ rm ${NAME}.gpkg

# create a subset from the PostgreSQL GeoPackage
$ ./gpkg-pg_dump.py "dbname='gpkg' user='gpkg'" ${NAME} -srcwin 3 3 1 1
```

Finally, drop the PostgreSQL-GeoPackage from the PostgreSQL database:

```
$ NAME=Sample-GeoPackage_Sentinel-2_Vienna_Austria

# drop the PostgreSQL GeoPackage
$ ./gpkg-pg_drop.py "dbname='gpkg' user='gpkg'" ${NAME}
```

8.2. Implementation Architecture

The implementation basically is made of four shell scripts, one holding SQL commands and three Python code:

- gpkg-pg_init.sql
- gpkg-pg_loadpkg.py
- gpkg-pg_dump.py
- gpkg-pg_drop.py

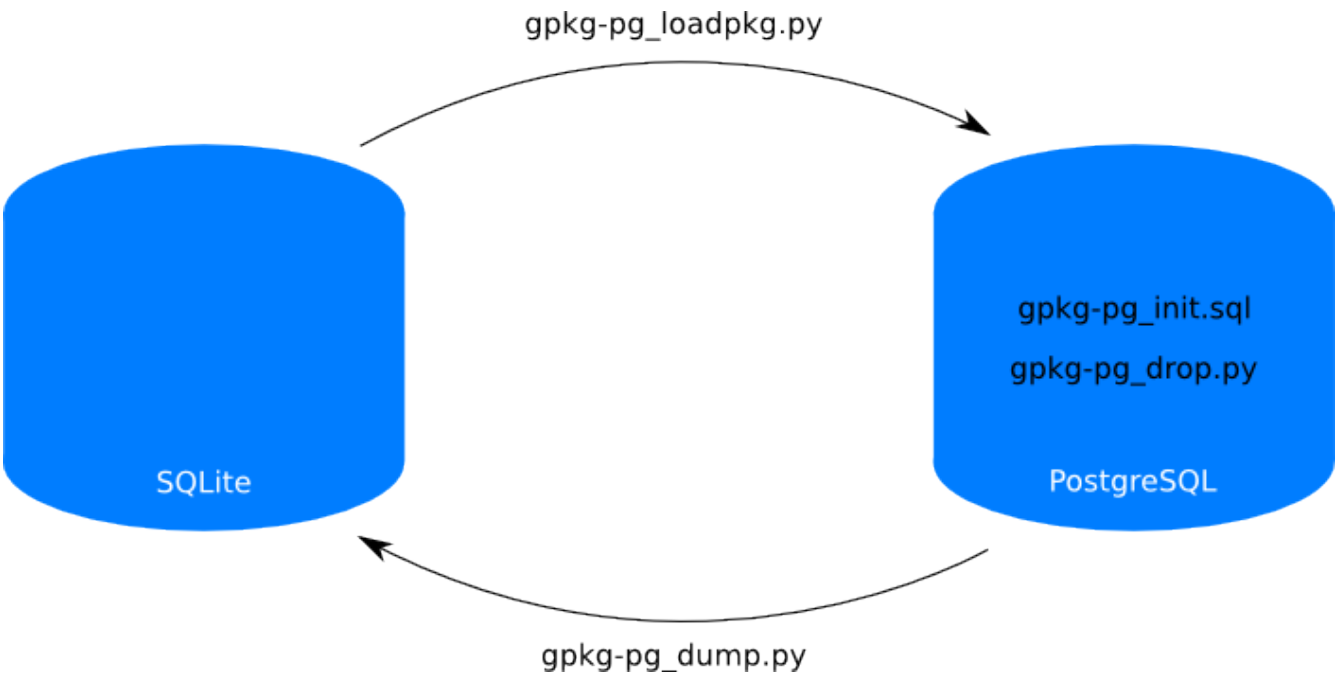

Figure 1. PostgreSQL-GeoPackage Architecture

In addition a Vagrant configuration is provided allowing for easy and quick evaluation of the

scripts.

8.2.1. gpkg-pg_init.sql

gpkg-pg_init.sql is used to initialize a PostgreSQL-GeoPackage. This means to load the table schema as specified by the GeoPackage standard or at least as close to the standard as possible into a new PostgreSQL-GeoPackage database.

Usage:

```
$ psql <PostgreSQL connection parameters> -f gpkg-pg_init.sql
```

This script might be used to generate a template database for usage in database creation like createdb -T template_gpkg my_spatial_db. This is similar to older versions of PostGIS.

Although the table schema had to be adjusted at some places, it was possible to copy the table schema and prove the suitability via successful data round trips i.e., loading SQLite GeoPackages, dumping them again, and proving that there a no differences by comparing the original versions to the newly dumped ones.

The following table summarizes the necessary adjustments:

Table 2. Schema adjustments

SQLite data types	PostgreSQL data types
INTEGER: size automatically adjusted between 1 and 8 bytes	BIGINT: always 8 bytes
DATETIME: with affinity to NUMERIC; DEFAULT (strftime('%Y-%m-%dT%H:%M:%fZ','now'))	TIMESTAMPTZ: timestamp with time zone 8 bytes; DEFAULT now()
REAL: 8-byte IEEE floating point number	NUMERIC: variable size (up to 131072 digits before and 16383 digits after the decimal point)
TRIGGER: direct definition	TRIGGER: using a slightly more complicate FUNCTION as shown in the example below

Particularly the handling of datetime elements is a little different in PostgreSQL than in SQLite. In SQLite column types are more or less just recommendations [16] whereas in PostgreSQL they are followed strictly supporting data type specific functions like exact datetime comparison including timezones. For this reason the datetime formatting specified in the GeoPackage standard is irrelevant in PostgreSQL but absolutely necessary in SQLite.

Another adjustment necessary is with respect to triggers which are defined quite differently. In SQLite triggers can be defined more directly whereas in PostgreSQL a function has to be defined first that can than be used in triggers. The example below shows the difference by first showing the PostgreSQL way followed by the SQLite way of defining a trigger:

```
# PostgreSQL
CREATE FUNCTION gpkg_tile_matrix_zoom_level_insert() RETURNS trigger
AS $gpkg_tile_matrix_zoom_level_insert$
    BEGIN
        IF NEW.zoom_level < 0 THEN
            RAISE EXCEPTION 'insert on table ''gpkg_tile_matrix'' violates constraint:
zoom_level cannot be less than 0';
        END IF;
        RETURN NEW;
    END;
$gpkg_tile_matrix_zoom_level_insert$ LANGUAGE plpgsql;
CREATE TRIGGER gpkg_tile_matrix_zoom_level_insert
BEFORE INSERT ON gpkg_tile_matrix
FOR EACH ROW EXECUTE PROCEDURE gpkg_tile_matrix_zoom_level_insert();

# SQLite
CREATE TRIGGER 'gpkg_tile_matrix_zoom_level_insert'
BEFORE INSERT ON 'gpkg_tile_matrix'
FOR EACH ROW BEGIN
SELECT RAISE(ABORT, 'insert on table ''gpkg_tile_matrix'' violates constraint:
zoom_level cannot be less than 0')
WHERE (NEW.zoom_level < 0);
END
```

Besides these adjustments the schema could be successfully copied directly to PostgreSQL and, as said above, proven to be suitable by showing that there are no differences between original SQLite GeoPackages and loaded and dumped PostgreSQL ones.

8.2.2. gpkg-pg_loadpkg.py

This script loads a SQLite GeoPackage into a PostgreSQL-GeoPackage database.

gpkg-pg_loadpkg.py takes an entire provided SQLite GeoPackage containing raster tile data only and loads it into the given PostgreSQL-GeoPackage database.

Usage:

```
$ ./gpkg-pg_loadpkg.py <SQLite GeoPackage filename> <PostgreSQL connection string>
```

This script might get a switch to make a selection of the data to be loaded, for example based on a spatial bounding box.

8.2.3. gpkg-pg_dump.py

This script dumps an entire PostgreSQL-GeoPackage database or spatial subsets thereof into a SQLite GeoPackage.

Usage:

```
$ ./gpkg-pg_dump.py <PostgreSQL connection string> <GeoPackage name> [-srcwin <xoff>
<yoff> <xsize> <ysize>]
```

The GeoPackage name is used to determine the table in which the tile data is stored as well as for the filename of the SQLite GeoPackage to generate.

In order to dump a spatial subset the `-srcwin` parameter may be used. It expects four integer values specifying a subwindow for dumping based on tile indexes starting from 0 0 at the top left.

8.2.4. `gpkg-pg_drop.py`

This script drops a PostgreSQL-GeoPackage from a database reversing the loading by the `gpkg-pg_loadpkg.py` script.

Usage:

```
$ ./gpkg-pg_drop.py <PostgreSQL connection string> <GeoPackage name>
```

8.3. Acknowledgment

The sample SQLite GeoPackage provided with the scripts was created from Sentinel-2 [17] data using GDAL [18].

Legal notice: Contains modified Copernicus Sentinel data [2016] [19].

8.4. Input

The scripts are able to load and dump the raster tiles from any SQLite GeoPackage. Typical data are base imagery or raster digital maps i.e., any RGBA coverages.

8.5. Output

Tile data stored in PostgreSQL using a table schema as defined by the GeoPackage standard or at least as close to the standard as possible as described above. The GeoPackage standard defines a number of general tables that describe the data like the table `gpkg_spatial_ref_sys` holding all reference system related information or `gpkg_contents` describing the data included in the GeoPackage. For tile data the `gpkg_contents` column of the `gpkg_contents` table has the value `tiles` and the `table_name` column points to the table holding the tile data as binary data. Two additional description tables for tile data are defined which are `gpkg_tile_matrix_set` and `gpkg_tile_matrix` defining the tile pyramid extent. The tile data itself is stored in a table of its own for each dataset following again a GeoPackage schema defining the columns `id`, `zoom_level`, `tile_column`, `tile_row`, and `tile_data`. The last column holds the binary data. Note that the JPEG or PNG tiles are stored as exactly the same binary data in both, SQLite and PostgreSQL.

The data can be accessed via standard PostgreSQL means as they are configured in the respective system. This might include Python bindings as used by the provided scripts as well as numerous

other bindings and direct SQL access.

The data or subsets thereof can also be dumped again as SQLite GeoPackage for further usage via the provided scripts.

8.6. Implementation Options Evaluation

The chosen implementation architecture stores the JPEG or PNG tiles as exactly the same binary data in both, SQLite and PostgreSQL. An alternate implementation option that has been evaluated but disregarded for the reasons provided below, is making use of the raster functionality provided by PostGIS.

The fact that software tools like `gdal_translate` and `raster2pgsql` are readily available to work with rasters in PostGIS is a big plus. Another benefit would be the support for simple querying and analysis of both rasters and vectors.

However, the raster data is stored in PostgreSQL as PostGIS Raster defined WKB [20]. This means that considerably more storage space is used compared to plain JPEG and PNG as used in GeoPackage. Additionally the data loading and dumping is way slower as there is always data conversion processing needed.

The table below shows some evaluation results using the same data and test setup as used for testing and evaluating the actual implementation in the next section.

Table 3. Implementation options evaluation results

Pixel size	Total number of tiles	SQLite GeoPackage file size (MB)	PostGIS average size on disk (MB)	% increase in size	Loading time average
2048 x 1024	42	0.37	6.86	1738.74%	00:00:00.84
4096 x 2048	170	1.16	22.12	1801.60%	00:00:02.38
8192 x 4096	682	3.98	77.10	1839.38%	00:00:09.50
16384 x 8192	2,730	13.85	260.61	1782.21%	00:00:37.43
32768 x 16384	10,922	62.03	1212.55	1854.78%	00:02:29.66
65536 x 32768	43,690	206.01	3836.50	1762.32%	00:10:01.96

The disk size needed to store the data in PostGIS is almost 20 times as the original SQLite GeoPackage file size.

As easily seen by comparing the results with the evaluation results table below, PostGIS is using way more disk space and the data loading is way slower although the time is even measured before applying constraints.

Given these evaluation results the PostGIS option has been disregarded and instead the implementation stores the exact same binary data for the JPEG or PNG tiles as the SQLite GeoPackage.

8.7. Testing & Evaluation

Two different tests have been performed on a number of SQLite GeoPackages.

The first kind of tests was to perform a full round trip with entire SQLite GeoPackages and check that the content of the SQLite GeoPackage did not change. This means running a `gpkg-pg_loadpkg.py` followed by a `gpkg-pg_dump.py` without using the `-srcwin` parameter and comparing the two SQLite GeoPackages.

Depending on the size of the SQLite GeoPackages the contents can either be dumped into text files using the command `sqlite3 <SQLite GeoPackage filename> .dump > text` and compared using `diff`. Alternatively or in addition both SQLite GeoPackages can be loaded in QGIS and visually compared.

Secondly the dumping of spatial subsets has been tested. Spatial subsets generated using the `gpkg-pg_dump.py` are compared to the same subsets generated using `gdal_translate`. Note that the `-srcwin` parameter in `gdal_translate` works in pixel space instead of the tile index one. In this case only the loading in QGIS and visual comparison can be used as `gdal_translate` regenerates the tiles which are thus not binary equal any more.

For the first kind of tests the commands below have been used to generate test data together with the following GDAL configuration file `gdal_wmts.xml`.

```
<GDAL_WMTS>
  <GetCapabilitiesUrl>
http://tiles.maps.eox.at/wmts/1.0.0/WMTSCapabilities.xml</GetCapabilitiesUrl>
  <Layer>terrain-light</Layer>
  <MaxConnections>12</MaxConnections>
  <ZeroBlockHttpCodes>404</ZeroBlockHttpCodes>
  <ZeroBlockOnServerException>true</ZeroBlockOnServerException>
</GDAL_WMTS>
```

```
$ gdal_translate gdal_wmts.xml -of GPKG -outsize 512 256 terrain-light_0.gpkg
$ gdal_translate gdal_wmts.xml -of GPKG -outsize 1024 512 terrain-light_1.gpkg
$ gdal_translate gdal_wmts.xml -of GPKG -outsize 2048 1024 terrain-light_2.gpkg
$ gdal_translate gdal_wmts.xml -of GPKG -outsize 4096 2048 terrain-light_3.gpkg
$ gdal_translate gdal_wmts.xml -of GPKG -outsize 8192 4096 terrain-light_4.gpkg
$ gdal_translate gdal_wmts.xml -of GPKG -outsize 16384 8192 terrain-light_5.gpkg
$ gdal_translate gdal_wmts.xml -of GPKG -outsize 32768 16384 terrain-light_6.gpkg
$ gdal_translate gdal_wmts.xml -of GPKG -outsize 65536 32768 terrain-light_7.gpkg
$ gdal_translate gdal_wmts.xml -of GPKG -outsize 131072 65536 terrain-light_8.gpkg
$ gdal_translate gdal_wmts.xml -of GPKG -outsize 262144 131072 terrain-light_9.gpkg
$ gdal_translate gdal_wmts.xml -of GPKG -outsize 524288 262144 terrain-light_10.gpkg
$ gdaladdo -r cubic terrain-light_1.gpkg 2
$ gdaladdo -r cubic terrain-light_2.gpkg 2 4
$ gdaladdo -r cubic terrain-light_3.gpkg 2 4 8
$ gdaladdo -r cubic terrain-light_4.gpkg 2 4 8 16
$ gdaladdo -r cubic terrain-light_5.gpkg 2 4 8 16 32
$ gdaladdo -r cubic terrain-light_6.gpkg 2 4 8 16 32 64
$ gdaladdo -r cubic terrain-light_7.gpkg 2 4 8 16 32 64 128
$ gdaladdo -r cubic terrain-light_8.gpkg 2 4 8 16 32 64 128 256
$ gdaladdo -r cubic terrain-light_9.gpkg 2 4 8 16 32 64 128 256 512
$ gdaladdo -r cubic terrain-light_10.gpkg 2 4 8 16 32 64 128 256 512 1024
```

The test data comprises a number of growing SQLite GeoPackages covering the whole Earth in EPSG:4326 projection starting from two tiles of 256 x 256 pixels and continuously adding a zoom level with double pixels in each axis.

The commands below have been repeatedly run multiple times to get reliable numbers. The duration numbers reported in the table below are average numbers of at least 5 runs.

```
$ vagrant ssh
$ sudo -s
$ cd /home/vagrant/PostgreSQL-GeoPackage/testdata/terrain-light/
$ aptitude purge postgresql postgresql-9.5 postgresql-9.5-postgis-2.2 postgresql-9.5-
postgis-scripts postgresql-client-9.5 postgresql-client-common postgresql-common $
postgresql-contrib-9.5
$ rm -rf /var/log/postgresql/ /var/lib/postgresql/ /etc/postgresql/
$ /bin/sh /home/vagrant/PostgreSQL-GeoPackage/vagrant/scripts/packages.sh
$ /bin/sh /home/vagrant/PostgreSQL-GeoPackage/vagrant/scripts/postgres.sh
$ for i in {0..9}; do echo ${i} && createdb -E UTF8 -U gpkg gpkg${i} && psql -U gpkg
gpkg${i} -f ../../gpkg-pg_init.sql && du -s /var/lib/postgresql/9.5/main/ &&
/usr/bin/time ../../$ gpkg-pg_loadpkg.py terrain-light_${i}.gpkg "dbname='gpkg${i}'
user='gpkg'" && du -s /var/lib/postgresql/9.5/main/; done
$ mkdir dumps
$ cd dumps/
$ for i in {0..4}; do echo ${i} && /usr/bin/time ../../../gpkg-pg_dump.py
"dbname='gpkg${i}' user='gpkg'" terrain-light_${i} && sqlite3 ../terrain-light_${i
}.gpkg .dump > ${i}b && $ sqlite3 terrain-light_${i}.gpkg .dump > ${i}a && diff ${i}b
${i}a > ${i}diff; done
$ for i in {5..9}; do echo ${i} && /usr/bin/time ../../../gpkg-pg_dump.py
"dbname='gpkg${i}' user='gpkg'" terrain-light_${i}; done
$ cd ..
$ rm -r dumps/
$ for i in {0..9}; do echo ${i} && du -s /var/lib/postgresql/9.5/main/ &&
/usr/bin/time ../../gpkg-pg_drop.py "dbname='gpkg${i}' user='gpkg'" terrain-light_${i}
&& du -s /var/lib/$ postgresql/9.5/main/; done
$ exit
```

The evaluation was run in a virtual machine as configured in the Vagrant configuration. As host a standard office laptop (Lenovo W540) with 16GB memory and a 2.80GHz 4 core Intel i7-4810MQ CPU was used. Note that time and other performance measurements were taken on the same platform at least slightly influencing the results. The following table summarizes the evaluation results:

Table 4. Evaluation results

Pixel size	Total number of tiles	SQLite GeoPackage file size (MB)	PostgreSQL-GeoPackage average size on disk (MB)	% increase in size	Loading time average	Loading throughput (MB/sec)	Loading throughput (tiles/sec)	Dumping time average	Dumping throughput (MB/sec)	Dumping throughput (tiles/sec)
2048 x 1024	42	0.37	0.55	46.60%	00:00:00.21	1.81	203.23	00:00:00.21	1.78	200.00
4096 x 2048	170	1.16	1.45	24.27%	00:00:00.43	2.68	392.31	00:00:00.29	4.01	586.21

Pixel size	Total number of tiles	SQLite GeoPackage file size (MB)	PostgreSQL-GeoPackage average size on disk (MB)	% increase in size	Loading time average	Loading throughput (MB/sec)	Loading throughput (tiles/sec)	Dumping time average	Dumping throughput (MB/sec)	Dumping throughput (tiles/sec)
8192 x 4096	682	3.98	4.59	15.55%	00:00:01.22	3.26	559.02	00:00:00.75	5.28	906.31
16384 x 8192	2,730	13.85	15.41	11.27%	00:00:04.72	2.93	578.39	00:00:01.74	7.98	1,573.49
32768 x 16384	10,922	62.03	116.15	87.24%	00:00:14.17	4.38	770.78	00:00:05.87	10.57	1,860.65
65536 x 32768	43,690	206.01	369.30	79.27%	00:00:54.69	3.77	798.94	00:00:19.16	10.75	2,279.97
131072 x 65536	174,762	699.60	1282.63	83.34%	00:03:06.18	3.76	938.67	00:01:15.55	9.26	2,313.12
262144 x 131072	699,050	2533.23	2852.59	12.61%	00:12:17.78	3.43	947.51	00:07:39.85	5.51	1,520.19
524288 x 262144	2,796,202	9465.21	10429.24	10.18%	00:48:34.42	3.25	959.44	00:39:55.14	3.95	1,167.45

The smaller GeoPackagaes have been compared after being dumped into text files whereas the bigger ones have only been compared by file size and visually using QGIS as described above. Only minor differences resulting from different floating point number calculations as well as different tile ordering have been observed.

8.8. Conclusions & Key Findings

This section highlights interesting observations and key findings from the evaluation results as shown in the table above.

The first interesting observation is the quite different increase in size from SQLite file size to PostgreSQL size on disk varying from just 10% to above 80%. The suspected reason is a suboptimal page size of typically 8KB in PostgreSQL. Note that the average tile size drops from 7KB to just 3.5KB for the bigger GeoPackages and the increase drops dramatically when the average tile size drops below 4KB i.e., half the page size.

The throughput of loading tiles into PostgreSQL-GeoPackages is quite constant with about 3.2-3.7 MB/sec or about 950 tiles/sec. The throughput of dumping tiles, however, is quite spread ranging from peak 10.7 MB/sec or well above 2000 tiles/sec to close to 4 MB/sec or around 1200 tiles/sec. This performance degradation is simply explained by I/O performance. Smaller GeoPackages fit entirely into memory whereas bigger ones not only need I/O for writing but also for reading. This can be

easily seen by the output of the `time` command which shows file system inputs and major page faults (no swaps though) for the bigger GeoPackages but not for the smaller ones.

In conclusion, the loading and dumping performance seems to be limited by I/O performance rather than any processing capabilities. The main task simply is moving data around on hard disk.

Chapter 9. Array Databases

SQL has been lingua franca for any-size data services in business, and has been tremendously successful in delivering flexible, scalable technology over decades. Also for vector data SQL is routinely used today, also thanks to ISO SQL/MM implementing a "Simple Features" model. This is in contrast to gridded, rasterized data - although they contribute massively to the Big Data deluge they are still maintained in file-based silos whose walls keep flexible retrieval and processing outside. This has contributed much to the historical divide between "data" (large, constrained to download, no search) and "metadata" (small, agile, searchable).

This is changing. A new class of NoSQL database systems, Array Databases [13][1][2][8], has set out to close this gap by providing declarative query support for massive multi-dimensional arrays while applying heavy optimization, parallelization, distributed processing, exploitation of heterogeneous hardware, and further mechamiss to achieve high performance and scalability.

We introduce concept and technology of Array Databases by way of the rasdaman ("raster data manager") system which effectively has pioneered this domain (cf. History).

9.1. Concepts

Multi-dimensional arrays constitute first-class citizens in rasdaman. An array has some *dimension* $d>0$, and along each dimension it has an *extent* given by integer coordinates with a lower and upper bound (geo coordinates, CRSs, and irregular grids are concepts added by a geosemantics layer implementing OGC W*S standards - see Figure 2). At each coordinate, a *cell* sits which has some value; all cell values in a given array pertain to some *cell type* which can be atomic or composite. Arrays are typed along their main characteristics: dimension, extent, and cell type. This allows to distinguish, for example, 2-D panchromatic, RGB, and hyperspectral imagery, elevation, and land use data.

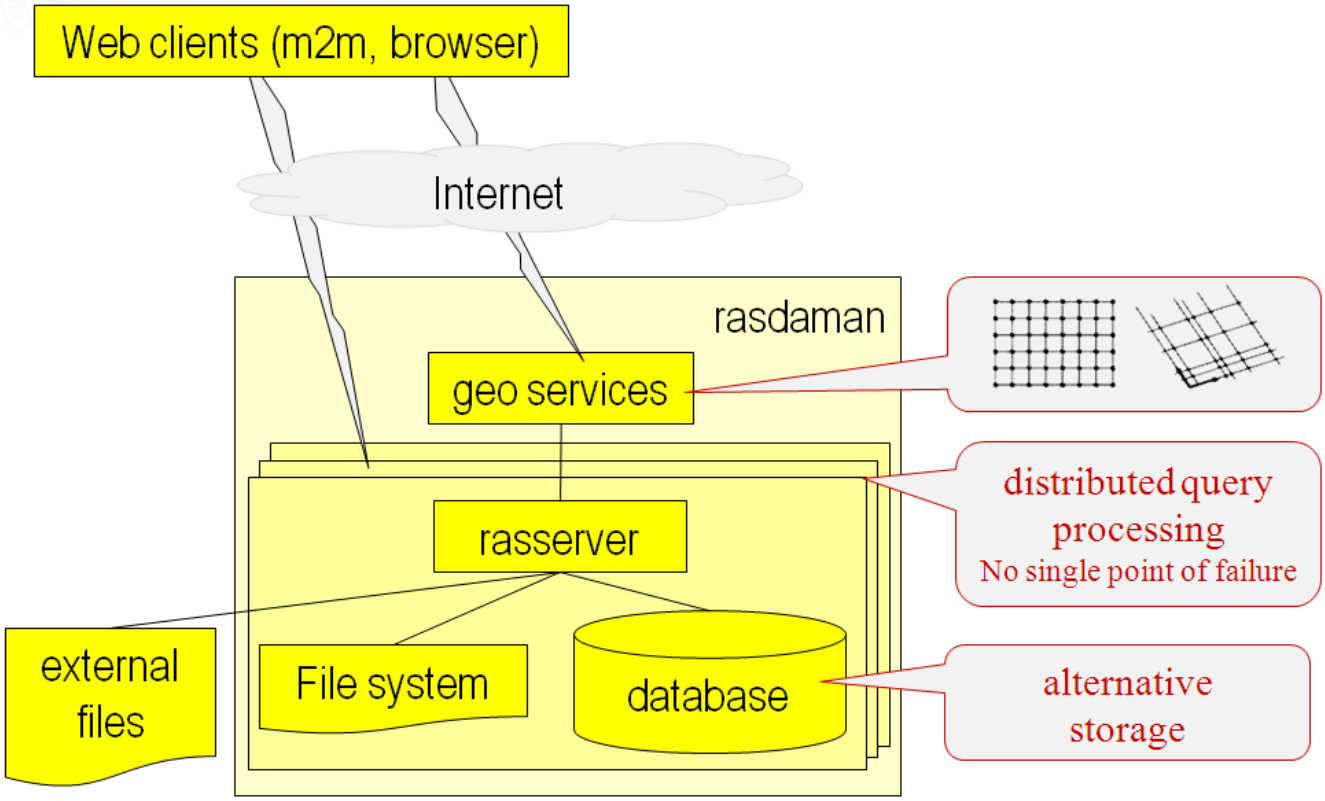

Figure 2. rasdaman architecture with array engine and geo semantics layer

On such arrays, expressions allow to retrieve sub-arrays through trimming and slicing (as in OGC WCS), processing of pixels (such as deriving the NDVI), timeseries analysis, and general signal/image processing andstatistics operations. For example, in rasdaman all WMS, WCS, and WCPS requests are internally mapped to array queries.

This demonstrates a major use case for the query language (see Figure 3): "Array SQL" is not meant to be an end-user language (except for experts), but a convenient client/server interface for tools that allow users to remain in their convenience zone, such as Leaflet, QGIS, python, or NASA WorldWind. Further, within the server it can act as an integration framework for comprehensive services, such as enhancing raster capabilities of MapServer through rasdaman.

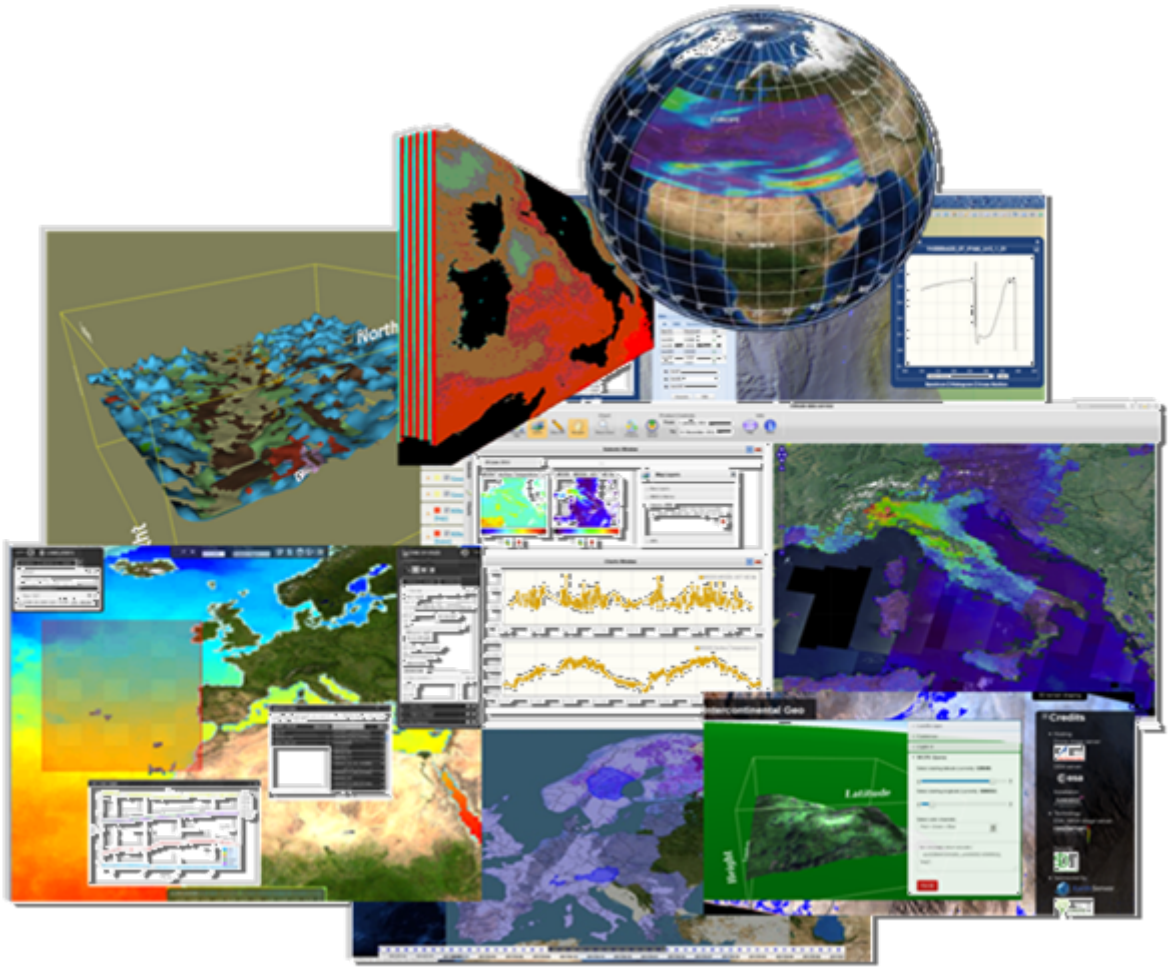

Figure 3. Sample array query results (source: rasdaman)

9.2. Storage and Tiling

There is common consensus among Array Database researchers that arrays need to be partitioned for storage. Often, such partitions are of equal size to ease implementation (called "chunking" in SciDB and others). More general is the tiling approach where each partition can have an individual size and shape. A general investigation on multidimensional tiling has been accomplished by Paula Furtado [7]. She classifies partitioning as follows (see Figure 4):

- aligned tiling, which can be regular (corresponding to chunking) or irregular;
- partially or totally nonaligned tiling.

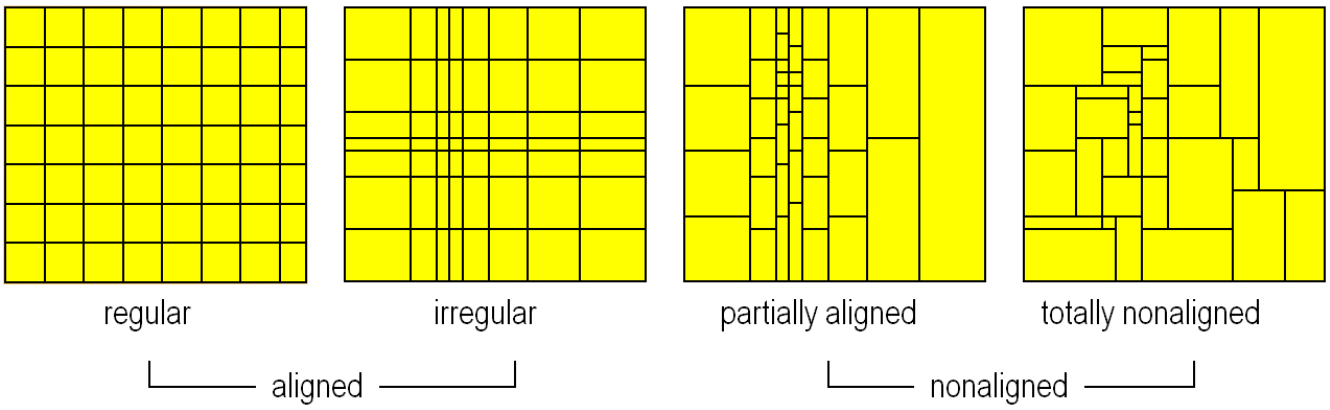

Figure 4. Partitioned array storage organization

As this universe of options turned out hard to handle by administrators (in particular with 3D+ situations), Furtado developed several strategies which make modeling easier. Among these are (cf. Figure 5): - regular tiling: this can be applied whenever no particular knowledge exists on the access patterns (i.e., the "query workload") - directional tiling: tiles can be stretched along dimensions by some ratio; for example, timeseries access can be optimized this way - area of interest: in this strategy, the user lists a number of areas which need to be accessible particularly fast; for all the rest of the array the system will do "something meaningful".

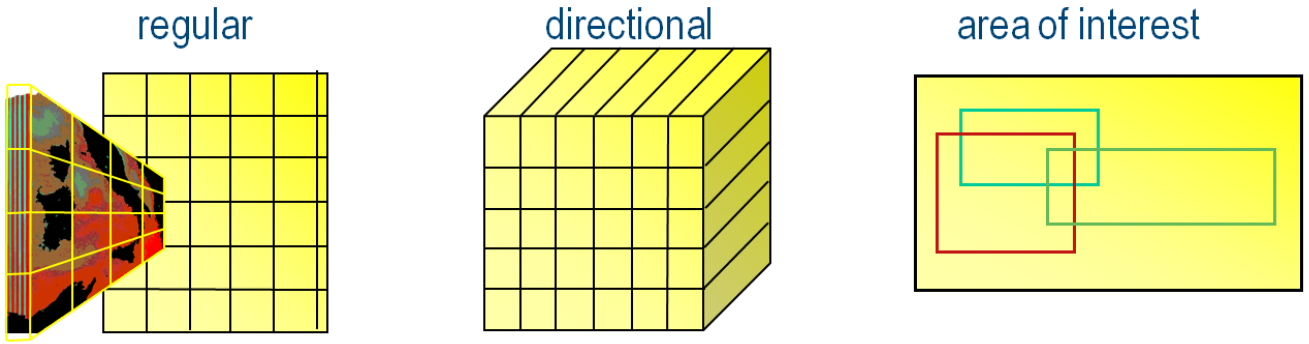

Figure 5. Selection of tiling strategies provided by rasdaman

In rasdaman, these strategies are available as part of the physical database design through a storage layout sublanguage which is attached to the INSERT statement. For example, the following statement inserts an array and, at the same time, fixes layout to some particular structure, index, and storage strategy:

```
insert into MyCollection
values ...
tiling area of interest [0:20,0:40], [45:80,80:85]
       tile size 1000000
       index d_index
       storage array compression zlib
```

Optionally, tiles can be compressed individually using either lossless or lossy methods; in the example above, zlib is specified.

9.3. Processing

Some array operations - concretely: "local operations" in Tomlin's categorization - are pleasingly parallelizable, something which Array DBMSs exploit massively when dispatching tiles across cores or within a cloud. However, there is more to it: graphics processing units (GPUs), if present, can support in certain class of operations, although not in all.

Even more potential has distributed processing of queries (see Figure 6). In this approach, incoming queries are analyzed individually to find out best split points. Next, subqueries as large as possible are generated and shipped to the nodes that hold the particular data item to be accessed and processed. One criterion for determining the split points is where minimal data transport takes place. Single queries have been split successfully over more than 1,000 cloud nodes [6].

Joins between arrays - occurring, for example, in overlays, matrix multiplication, terrain draping - pose particular challenges and require special algorithms [4].

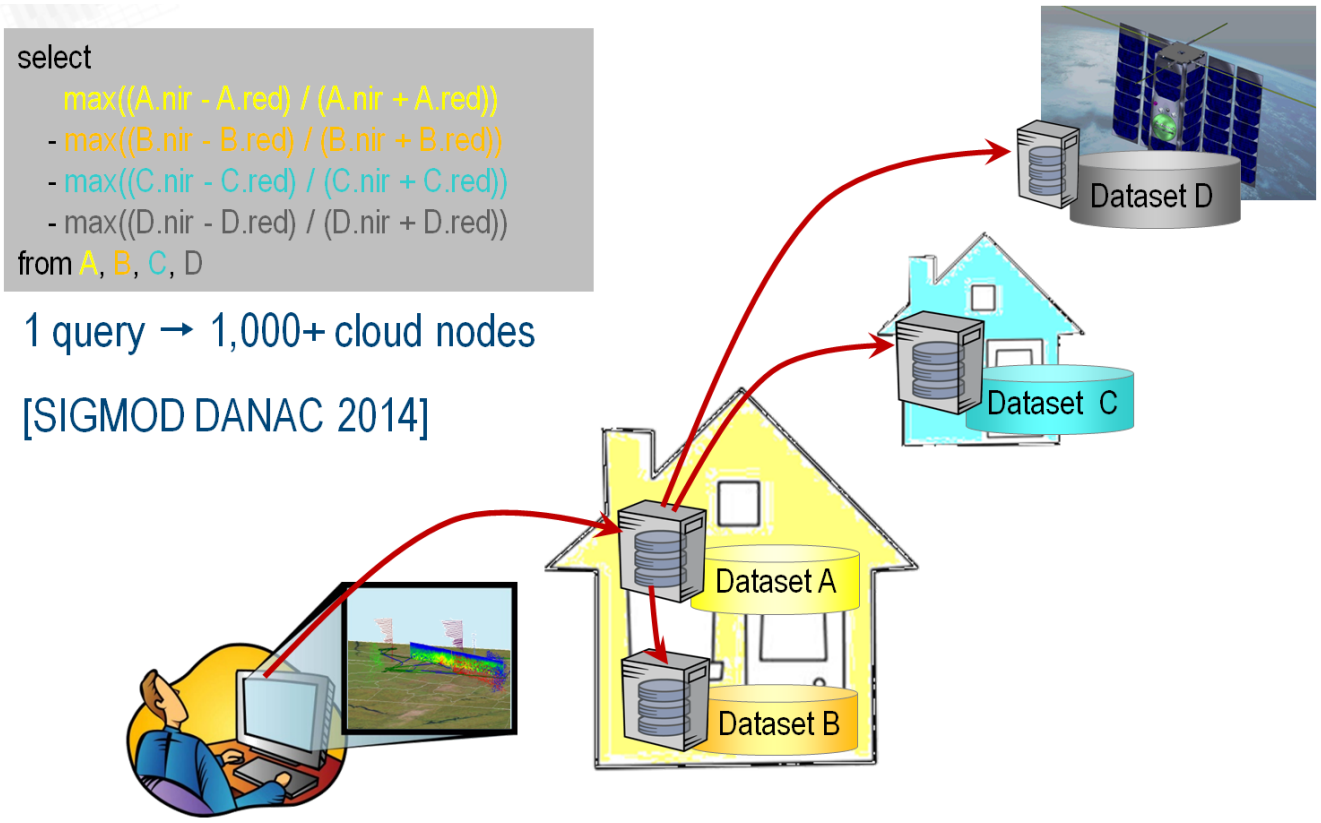

Figure 6. Federated inter-continental array query processing

9.4. History

Figure 7 gives an overview on the history of Array Databases. The historically first Array DBMS was rasdaman [1][2]. Several early approaches addressing arrays have been discontinued (such as Paradise). However, meantime an increasing takeup of the concept can be observed with array systems including TerraLib, SciQL, SciDB, Ophidia, EXTASCID, and several more. The figure below illustrates this development.

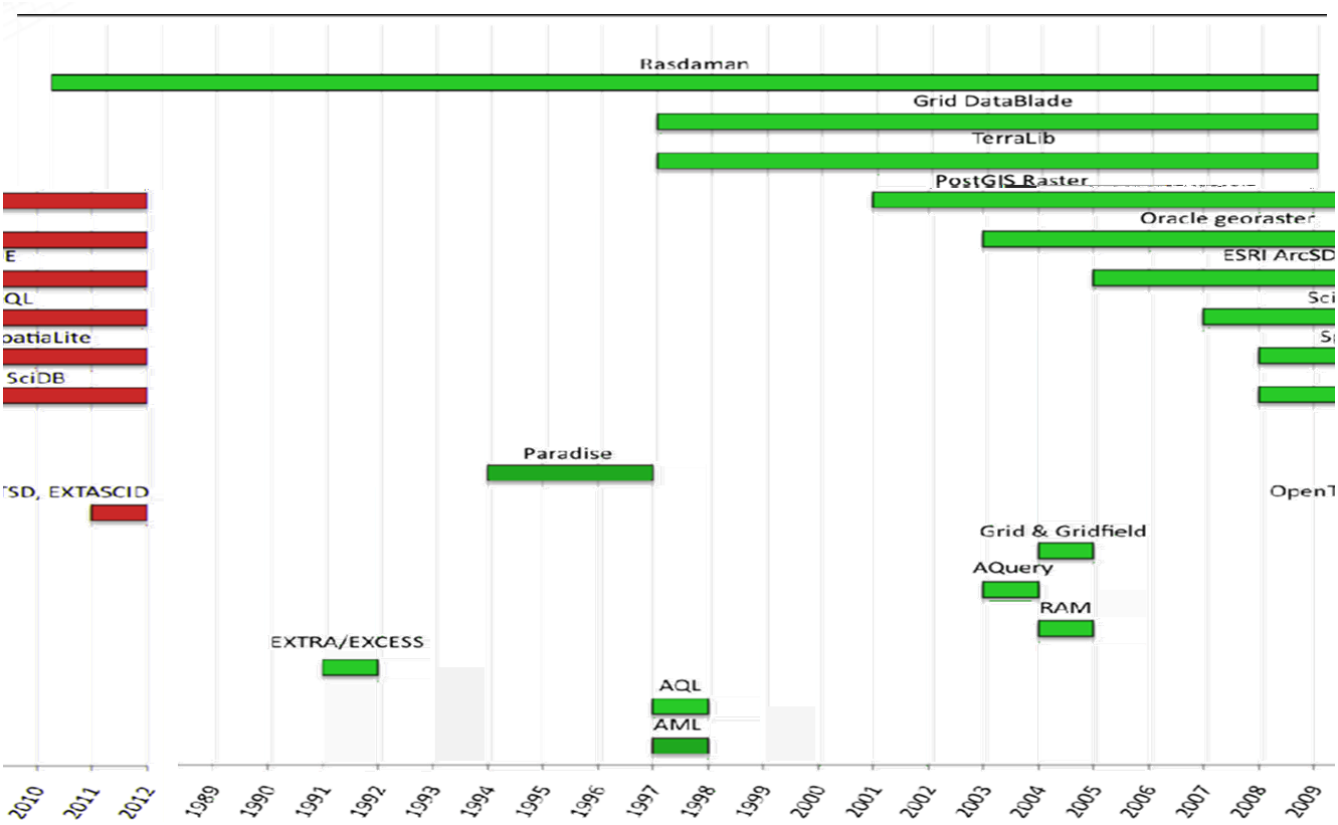

Figure 7. Timeline of Array Database technology

9.5. Standards

Array services have reached a degree of maturity meantime which makes them suitable for standardization. On domain-independent level, ISO is enhancing its SQL query language with any-size multi-dimensional arrays as a new attribute type in tables, together with a set of declarative operators, based on the concepts of the rasdaman Array Algebra and query language. This standard, which will go by the official title ISO 9075 Part 15: SQL/MDA (MDA standing for "Multi-Dimensional Arrays") is expected to be a game changer in Big Science Data [11][9]. SQL/MDA can be expected to become the lingua franca for data access in and across data centers worldwide. Currently it is at Committee Draft (CD) status; expectation is that it will get adopted by summer 2017.

Specifically for spatio-temporal arrays, OGC provides a modular suite of specifications around the notion of coverages [10]. The abstract coverage data model is defined in OGC Abstract Topic 6 (identical to ISO 19123), its concrete, interoperable instantiation is the OGC Coverage Implementation Schema (CIS). Such coverages include spatio-temporal regular and irregular grids, point clouds, and general meshes. The corresponding service model is provided by the OGC Web Coverage Service (WCS) suite which offers functionality ranging from simple subsetting - defined in WCS Core - up to ad-hoc spatio-temporal analytics with OGC Web Coverage Processing Service (WCPS) [3]. All these WCS facets are implemented in rasdaman, and in operational use since several years.

ISO has started adoption of OGC CIS and plans to follow on with OGC WCS. INSPIRE, the European legal framework for a common Spatial Data Infrastructure (SDI), utilizes coverages for orthoimagery, elevation data, as well as ocean and atmospheric data; currently, it is adopting WCS as a so-called "Coverage Download Service", but already seeing the value of WCPS, e.g., for creating

visualizations.

In RDA, an Array Database Assessment Working Group (ADA:WG) has been established [12] which is conducting a neutral, thorough hands-on evaluation assessing available Array Database systems and comparable technology ...

- based on relevant standards, such as the NIST Big Data Reference Architecture, ISO “Array SQL” [11][9], and OGC Web Coverage Processing Service (WCPS) [3] for the geo domain;
- comparing technical criteria like functionality, thereby eliciting the state of the art;
- establishing and running a combination of domain-driven and domain-neutral benchmarks that will be run on each platform;
- as well as real-life, publicly accessible deployments at scale.

The outcome will consist of the ADA:WG report together with the open-source benchmarking software and the services established, will establish a hitherto non-existing overview on the state of the art and best use of Array Databases in science, engineering, and beyond.

9.6. Summary

Array Databases provide languages for flexible queries on multi-dimensional, spatio-temporal "datacubes", paired with technology which allows for highly effective server-side optimization, parallelization, distributed processing, and use of heterogeneous hardware. After many years of development, they are mature and in operational use on databases exceeding 100 Terabytes and with massive query parallelization.

In short, Array Databases provide the following benefits over ad-hoc crafted silo solutions: * flexibility: any query, anytime * performance and horizontal (cloud!) as well as vertical (GPUs!) scalability * information integration: common query space for data and metadata

In the context of OGC Web services, the "datacube" paradigm of coverages has proven useful for a simple, yet flexible handling of such Big Data. With the open-source rasdaman system, practice-proven technology is available for serving coverage massive datacubes conveniently through OGC standards including WMS, WCS, WCPS, WPS, and others. The rasdaman system is available in open source from http://www.rasdaman.org and can be deployed from source, as RPMs and Debian packages, and as ready-made virtual machines.

Maturity and scalability of the Array Database concept shows in practical use. An online demo [5] showcases WCS and WCPS, provided by rasdaman, in 1-D through 5-D use cases including visual interaction as well as query sandboxes. The EarthServer initiative (http://www.earthserver.eu), a collaboration between Europe, US, and Australia is using rasdaman for establishing a datacube federation on 3-D x/y/t satellite image timeseries and x/y/z/t weather data. Among the participating data centers are European Space Agency (ESA), European Centre for Medium-Range Weather Forecasts (ECMWF), Plymouth Marine Laboratory, and National Computational Infrastructure (NCI) Australia. Currently, rasdaman guarded data holdings exceed 145 Terabytes per database, the Petabyte frontier is planned to be crossed early 2017. All in all, users will get provided with a single common information space on datacubes which can be combined in a "mix and match" style anytime.

Chapter 10. Sensor Web Enablement and Big Observation-Databases

More and more Sensor Web Enablement (SWE) components are deployed in different domains such as hydrology, oceanography or air quality in order to make observation data accessible via the Web. However, besides variability of data formats and protocols in environmental applications, the fast growing volume of data with high temporal and spatial resolution is imposing new challenges for Sensor Web technologies when sharing observation data and meta data about sensors.

Variability, volume and velocity are the core issues that are addressed by Big Data concepts and technologies. Most solutions in the geospatial sector focus on remote sensing and raster data, whereas big in-situ observation data sets relying on vector features require novel approaches.

This chapter aims to rise the awareness of this gap. For this, the general concepts of the SWE framework will be introduced in section 10.1 and section 10.2 will discuss questions, divided in the three categories *Accessibility*, *Processing* and *Storage*, that need to be answered in order to deal with big data sets in infrastructures for observational data.

Section 11 will give an outlook how these questions might be answered.

10.1. Sensor Web Enablement

The Sensor Web Enablement suite of standards developed by the OGC aims to interoperably integrate sensors and their observation data into web-based (spatial) data infrastructures [14]. The following subsections introduce the core elements of the SWE architecture which comprises specifications for web service interfaces as well as data models and encodings.

10.1.1. Sensor Observation Service

The OGC Sensor Observation Service (SOS) interface standard is the most widely deployed web service of the SWE framework. It has reached version 2.0 which incorporates experiences gained during practical application of the first SOS version over several years.

The SOS interface allows pull-based access to observation data and sensor metadata. This means that the SOS acts as a mediator between clients and a measurement archive (e.g. database) or sensor system. It enables clients to query observation data of heterogeneous sources via a standardized interface. The SOS standard defines a set of parameterized operations and it relies on the data model/encoding standards of the SWE framework to provide standardized outputs.

The core operations of the SOS interface are:

`GetCapabilities`
: Retrieve metadata about a SOS server (e.g. supported operations and available data sets)

`DescribeSensor`
: Access metadata about the sensors or processes which have generated the observation data offered by the SOS server

`GetObservation`
: Retrieval of observation data/measurements

An important extension of the SOS interface is a group of transactional operations (`InsertSensor` and `InsertObservation`) for publishing new sensors and observations data on a SOS server

Another important operation is the `GetFeatureOfInterest` operation which allows the retrieval of the geometric features to which observations are associated. It provides the required spatial context, by serving e.g. point or polygon features of the feature that is being observed (see section Observation & Measurements).

10.1.2. Sensor Planning Service

The OGC Sensor Planning Service (SPS) interface standard offers functionality for controlling sensors and measurement processes. This means that the SPS is not suited for accessing observation data but to control the process how this data is generated.

Important operations of the SPS interface, which is also available in version 2.0, comprise:

`GetCapabilities`
: Retrieve metadata about a SOS server (e.g. supported operations, sensors which can be controlled, tasks that can be executed by the sensors)

`DescribeTasking`
: Access information on how to formulate tasking requests (i.e. required parameters and their types)

`GetFeasibility`
: Checking whether a task for a specific sensor is feasible or not (e.g. a sensor might be blocked at a specific point in time by another task)

`Submit`
: Send a tasking request to an SPS server

`GetStatus`
: Determining the status of the execution of a task

`DescribeResultAccess`
: Determine how the data collected as result of a task can be accessed (e.g. this operation might return a reference to a SOS server that provides the collected data)

10.1.3. Eventing

While the SOS offers pull-based data access (i.e. a request-response communication pattern) in some cases it is necessary to deliver observation data to a consumer as soon as the data is available (e.g. in alerting applications).

Such functionality requires push-based, asynchronous delivery of sensor data. Such an eventing mechanism is based on a publish/subscribe communication pattern: A consumer subscribes for a notification, the eventing component analyses incoming sensor data, and forwards the new (relevant) observations to the subscriber in (near) real-time.

Within the OGC SWE framework, there is not yet a corresponding adopted standard available. However there are several specifications available:

- OGC Sensor Alert Service (SAS): This specification is rather old and was published as an OGC Best Practice Paper. However it has some drawbacks because it is relatively tightly coupled to a specific communication protocol (XMPP) and it does not support complex event processing concepts for detecting relevant events.
- Sensor Event Service (SES): This specification (not released as an official standard, but available as discussion paper) can be considered as successor of the SAS: It supports the definition of complex event patterns that shall be detected. For defining these rules the SES relies on the OGC Event Pattern Markup Language (EML) Discussion Paper. However, the SES has not been advanced to an official standard.
- OGC Pub/Sub: This standard has been adopted in 2016 and it offers a specification how to implement publish/subscribe communication for OGC Web services. Thus, this standard should be considered as basis for implementing eventing functionality. However, the OGC Pub/Sub standard goes beyond the SWE framework and it does not offer further details on how to define patterns for event subscriptions. This needs to be specified as an addition.
- OGC Web Processing Service (WPS): In the OGC IMIS IoT Pilot (2015-2016) an event processing profile of the Web Processing known as Web Event Processing Service or WEPS, was discussed. It can be expected that this profile together with the OGC Pub/Sub standard may help to build interoperable eventing applications.

10.1.4. SensorML

While the previous three specifications define Web service interfaces for sensor related functionality, the OGC Sensor Model Language (SensorML) 2.0 offers a data model and XML encoding for metadata about sensors and measurement processes (thus, the SOS uses SensorML as a response format of the `DescribeSensor` operation). Measurement processes can range from singular sensor stations to complex sensor platforms that form a system which also describes measurement processing steps in high detail (e.g. that and how a normalization process is applied).

Typical elements which may be included in a SensorML document comprise for example:

- Keywords characterizing a sensor
- Identifiers (e.g. serial numbers, sensor names and ids)
- Classifiers (e.g. information about application domains of a sensor)
- Characteristics (e.g. size, weight of a sensor)
- Capabilities (e.g. resolution, sampling rate, etc. of a sensor)
- Valid time (information for which time period a sensor description is valid)
- Input and outputs of a sensor or measurement process
- Contact information (e.g. of the sensor operator, responsible scientist, manufacturer)

The SensorML standard has intentionally been defined in an application and domain independent manner. This means that only a very small set of mandatory information is provided in all SensorML documents. To increase interoperability between different communities, it is important

to develop sensor profiles with additional restrictions and requirements that specify that certain information will be provided in a certain way.

10.1.5. Observation & Measurements

Complementary to the metadata model and encoding of SensorML 2.0, the Observation and Measurements (O&M) 2.0 standard offers a model and an XML encoding for data observed by sensors (archived/delayed-mode as well as real-time data). For O&M it is important to state that the data model has been adopted as an ISO standard while the XML encoding is an OGC standard.

Typical information required for an observation conforming to the O&M standard comprises:

- Process: The sensor, model or algorithm which has delivered the observed value
- Observed Property: The parameter which is observed (e.g. water temperature)
- Feature of Interest: The geographic feature to which the observed value is associated
- Observation Result: The measured value itself (including unit of measurement, if applicable)
- Time Stamps: Different time stamps relevant to the observation

10.2. Challenges of SWE and Big Data

While Eventing, Planning and Sensor metadata descriptions do not raise issues for big datasets, the Sensor Observation Service as an possible gateway for a long term observation archive and O&M as an encoding of these observations may impose limitations upon the use of SWE framework components. The following presents question and challenges that need to be answered and resolved in order to deal with large observational data sets in Sensor Web applications.

How can views on big data sets and derived information products be made accessible in the Sensor Web?

- What are typical request scenarios of observation data for search, download, visualization and processing?
- Are current Sensor Web standards capable of and suitable for handling massive observation data sets?
- Which service standards, especially SOS, WCS, WCPS, WPS, are appropriate for search, download, visualization and processing?
- Which conceptual models and encodings, e.g. O&M or NetCDF, are appropriate for data transfer?

How can big observation data sets be processed efficiently and how does the underlying storage structure influence performance?

- How does the WPS handle situations in which transferring datasets is hard to achieve? Can the WPS be used as a Rich-Data-Interface for Big Data observation databases?
- How can predefined, parameterized or even interactive analyses be realized?
- How could a query language that enables on-demand analysis of time series data, similar to

what the WCPS Language does for coverages, look like?

- The combined analysis of multiple datasets of different origins offers the possibility of interdisciplinary research. How can this be accomplished with such high volumes of data?

How can big heterogeneous spatio-temporal datasets be organized, managed, and provided to Sensor Web applications?

- Currently, data is for the most part organized as files (e.g. NetCDF or CSV/TSV) each containing a a fixed time interval of multiple sensors. This allows for fast insertion times, but makes queries comprising a single series over an extended amount of time expensive.
- How can queries in both request dimensions, i.e. requesting a subset of a timeseries vs. requesting the measurements of a phenomenon across multiple sensors at a single point in time, be realized and how can this be accomplished while retaining fast insertion times of near real time data?
- Are existing database technologies, e.g. distributed, array and object databases applicable?
- Does there exist a solution that offers acceptable trade offs between the different requirements?

Chapter 11. Recommendations

The main recommendation for future work is to integrate PostgreSQL-GeoPackages in an OGC service like WPS or WMTS and to compare it to the same service relying on SQLite GeoPackage as data input format. Furthermore, we recommend to provide an abstract model for GeoPackages which is independent of the underlying database technology (Section 8).

Section 10 raises many challenges regarding the application of Sensor Web infrastructures in environments dealing with large heterogeneous observational data archives. It is recommended to develop solutions addressing these challenges and to create best practices on how to handle these data sets.

Appendix A: Revision History

Table 5. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
April 12, 2016	Christian Autermann	0.1	all	Initial ER
June 30, 2016	Christian Autermann	0.2	all	First full-draft ER
October 28, 2016	Christian Autermann	1.0	all	Draft ER
April 7, 2017	Christian Autermann	1.1	all	Draft ER

Appendix B: Bibliography

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

[2] P. Baumann: A Database Array Algebra for Spatio-Temporal Data and Beyond. The Fourth International Workshop on Next Generation Information Technologies and Systems (NGITS '99), July 5-7, 1999, Zikhron Yaakov, Israel, Springer LNCS 1649

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

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

[5] nn: Big Earth Data Standards - build your own spatio-temporal dataset. http://standards.rasdaman.com

[6] A. Dumitru, V. Merticariu, P. Baumann: Exploring Cloud Opportunities from an Array Database Perspective. Proc ACM SIGMOD Workshop on Data analytics in the Cloud (DanaC'2014), June 22 - 27, 2014, Snowbird, USA

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

[8] nn: Jacobs University Large-Scale Scientific Information Systems Research Group. https://www.jacobs-university.de/lsis

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

[10] OGC: Coverages Domain Working Group. http://external.opengeospatial.org/twiki_public/CoveragesDWG/WebHome

[11] ISO: SQL 9075 Part 15: MDA (Multi-Dimensional Arrays)

[12] RDA: Array Database Assessment Working Group: https://www.rd-alliance.org/groups/array-database-working-group.html

[13] Wikipedia: Array DBMSs. http://en.wikipedia.org/wiki/Array_DBMS

[14] Bröring, A., J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, R. Lemmens: New Generation Sensor Web Enablement. MDPI Sensors, vol. 11, 2011, pp. 2652-2699

[15] nn: PostgreSQL-GeoPackage http://github.com/EOX-A/PostgreSQL-GeoPackage

[16] nn: Vagrant instructions http://github.com/EOX-A/PostgreSQL-GeoPackage/tree/master/vagrant

[17] nn: Datatypes In SQLite Version 3 - Type Affinity https://www.sqlite.org/datatype3.html#type_affinity

[18] nn: Sentinel-2 https://sentinel.esa.int/web/sentinel/missions/sentinel-2

[19] nn: GDAL http://gdal.org

[20] nn: Sentinel Data Legal Notice https://scihub.copernicus.eu/twiki/pub/SciHubWebPortal/TermsConditions/Sentinel_Data_Legal_Notice.pdf

[21] nn: PostGIS Raster defined WKB https://trac.osgeo.org/postgis/wiki/WKTRaster/Documentation01