Open Access

Open Journal of Internet of Things (OJIOT)
Volume 8, Issue 1, 2022

www.ronpub.com/ojiot
ISSN 2364-7108

Space Cubes: Satellite On-Board Processing of Datacube Queries

Dimitar Misev, Peter Baumann

Jacobs University, Campus Ring 12, 28759 Bremen, Germany, {d.misev, p.baumann}@jacobs-university.de

ABSTRACT

Datacubes form an accepted cornerstone for analysis- and visualization-ready spatio-temporal data offerings. The increase in user friendliness is achieved by abstracting away from the zillions of files in provider-specific organization. Datacube query languages additionally establish actionable datacubes, enabling users to ask "any query, any time" with zero coding. However, typically datacube deployments are aiming at large scale, data center environments accommodating Big Data and massive parallel processing capabilities for achieving decent performance. In this contribution, we conversely report about a downscaling experiment. In the ORBiDANSE project a datacube engine, rasdaman, has been ported to a cubesat, ESA OPS-SAT, and is operational in space. Effectively, the satellite thereby becomes a datacube service offering the standards-based query capabilities of the OGC Web Coverage Processing (WCPS) geo datacube analytics language. We believe this will pave the way for on-board ad-hoc processing and filtering on Big EO Data, thereby unleashing them to a larger audience and in substantially shorter time.

TYPE OF PAPER AND KEYWORDS

Short Communication: *Datacubes, cubesat, satellite, array databases, rasdaman, SQL/MDA*

1 INTRODUCTION

Never before has it been so inexpensive to obtain large amounts of Earth observation satellite imagery, helping to monitor and understand our planet and its evolution in time. At the same time this brings many challenges [19].

Increases in spatial sensor resolution result in more detailed but also larger data acquisitions, and data download during ground station overpasses becomes a bottleneck [11] requiring complex scheduling techniques [7][14].

Further, data providers still tend to think in “archives” akin to Web file systems, which offer at most download of individual scenes as acquired by the satellite sensors. Such data organization tends to perform badly in timeseries processing and leaves it to the end user to deal with the data filtering, alignment, and indexing tasks required for any serious data mining and analytics.

Finally, data download, processing, and ingestion into archives takes significant time, so it is rarely viable for real-time applications.

These and further impediments call for shifting processing on board the satellites so that not raw data, but answers to user questions can be provided in near-real time. With increasingly more computing power

This paper is accepted at the International Workshop on Very Large Internet of Things (VLIoT 2022) in conjunction with the 2022 VLDB Conference in Sydney, Australia. The proceedings of VLIoT@VLDB 2022 are published in the Open Journal of Internet of Things (OJIOT) as special issue.

and more standardized software architectures available on-board [10][16] this vision appears feasible today.

This begs the question what high-level access and "answering user questions" could mean. Our understanding is driven by the notion of Analysis-Ready Data (ARD) [19] on the one hand and the concept of *coverages* as per the OGC and ISO standards [6][17][4] on the other hand. In combining both concepts the OGC Coverage Implementation Schema (CIS) [6] allows representing, among others, multi-dimensional spatio-temporal datacubes. The standard defines requirements such data need to fulfill to be ready for analysis – be homogeneous in that one single coordinate reference system governs the whole extent of it, for example. On this data model both a low-level access service called *Web Coverage Service* (WCS) [1] is defined, and a high-level datacube analytics language named OGC *Web Coverage Processing Service* (WCPS) standard [4][3], which is paralleled by the ISO SQL/MDA standard [13], an extension of standard SQL for management and querying of multidimensional arrays, just without space/time semantics to be domain agnostic. Both WCPS and SQL/MDA are suitable candidates for the high-level interfaces a satellite should provide to enable direct on-demand access and analytics.

Besides the elevated quality of service there is also a gain in performance suggested by the observation that the more concretely user requests can be directly answered on the server, the less data need to be delivered to the client, thereby minimizing download costs. One demonstration of this is available in the Earth Datacube Playground "race" between an Apache Web server and a rasdaman database service [17].

Based on these considerations we have proposed with the ORBiDANSe project, back in 2016, to conduct a proof of concept by equipping forthcoming ESA OPS-SAT [10] with the rasdaman datacube engine. The main goal was to allow WCPS and SQL/MDA queries to be sent to the satellite, receiving back the query evaluation results.

In this paper we report about the first experiments now that the satellite has become operational. The main contributions are (i) porting an existing, datacenter-proven datacube engine to a cubesat, (ii) optimizing this orbital data service by minimizing its resource footprint so as to run in the limited environment, and (iii) demonstrating feasibility in space.

The remainder of this paper is organized as follows. In Section 2 we inspect related work in the field. In Section 3 we present the on-board experiment of the datacube engine on the cubesat, followed by a discussion of the results in Section 4. Finally, Section 5 concludes the plot.

2 RELATED WORK

NASA SpaceCube is a hardware platform optimized for efficient on-board processing of any code [16]. As such, it supports neither datacubes nor datacube queries. In fact, rasdaman potentially could be ported to SpaceCube.

ESA Phi-Sat-1 is a recently launched cubesat where emphasis is put on bringing standard software architectures to space. Phi-Sat-1 concentrates on AI processing on board with a fixed functionality (such as cloud detection) determined by the pre-flight training [12], as opposed to the ORBiDANSe flexibility of "any question, any time". A combination of both approaches would be an interesting future project.

On the more theoretical side, the work on iSat explores how satellites can be transformed from fixed relay nodes into dynamic edge computing nodes capable of loading apps on board, essentially forming an IoT cloud in space, and interfacing with standard cloud services on the ground [20]. This idea is very similar in spirit to OPS-SAT, but goes a step further by considering satellite constellations. Simulated experiments on the iSat model demonstrated significant improvements to time and power consumption when executing tasks on board.

In summary, we are not aware of any activity to perform on-board processing (i) using a datacube approach and (ii) providing flexible query access from ground.

3 RASDAMAN @ OPS-SAT

In this section we give an overview of the OPS-SAT cubesat and the rasdaman Array DBMS aka datacube engine. After that, we report on our challenges encountered while porting rasdaman to the OPS-SAT environment.

3.1 The OPS-SAT Cubesat

OPS-SAT is a 3-unit cubesat of size 10 x 10 x 30 cm with deployable solar panels on each side [10] (Figure 1). The nano-satellite has been launched by ESA as an experimental platform for novel hardware and software concepts. It offers a range of resources on-board, including standard CPU / storage / memory, field-programmable gate arrays (FPGAs), camera, GPS, and an attitude determination and control system (ADCS). The Satellite Experimenter Processing Platform (SEPP) features a 925 MHz dual-core ARM Cortex A9 Hard Processor System (HPS), 16 GB of flash storage, 1 GB of DDR3 CPU RAM with ECC, and an integrated Cyclone 5 FPGA with 1 GB of dedicated DDR3 FPGA RAM, creating a high-bandwidth system

for embedded applications with parallel processing. The images acquired with the on-board camera are 2048x1944 pixels with standard visible RGB channels; an example is shown on Figure 2.

OPS-SAT supports deployment of external software on-board in the form of standardized application packages, which is a novel development in the satellite world. The NanoSat Mission Operations Framework software development kit (NMF SDK) helps experimenters in on-board software development by providing an API for utilizing the satellite's resources, such as camera, GPS, ADCS, etc. This system runs an embedded Linux operating system with support for executing Java 1.8 applications, native C/C++ executables compiled for the *armhf* architecture, Python 2.7 scripts, and a standard Posix shell with the *busybox* utilities.

A service called *Space Shell* is also provided to experimenters on ground for submitting Linux commands as unprivileged user in the on-board shell in real-time. Any output generated during command execution will be downlinked via S-Band. Further, an interactive ground application is available for the experimenter.

For our experiment setup, we developed a Java application that allows executing several actions remotely via the NMF SDK, including: start and stop rasdaman; take a picture and insert into rasdaman either as a 2D array or updating a 3D time-series datacube; and adjust the camera exposure to an automatically determined value.

3.2 The rasdaman Datacube Engine

The rasdaman ("raster data manager") datacube engine has pioneered Array Database Management Systems (DBMS) as a new class of NoSQL DBMSs [1][5][2]. Like a conventional DBMS rasdaman offers a query language, rasql, with specific operators on multi-dimensional datacubes, rather than conventional tables. Among others, rasdaman supports spatio-temporal queries in a genuinely multi-dimensional way, with all functionality available on every axis, including time.

In 2019, this high-level, declarative query language has been adopted as a further component of the ISO SQL standard under the name SQL / MDA (Multi-Dimensional Arrays) [13].

Architecturally, rasdaman resembles a fully-fledged DBMS with its array engine crafted as a full-stack implementation in fast C++. The multi-parallel worker processes operate on arbitrarily tiled arrays; the administrator can override the default tiling by specifying individual size and shape for storage; this resembles a tuning factor which, however, remains completely invisible to the query writer. Further tuning parameters include compression, indexing, cache sizing, etc. A number of highly effective optimizations are applied to each incoming query individually, such as query rewriting into a more efficient form, cost-based optimization, intelligent cache utilization, compilation into machine code, multi-core parallelization, distributed processing, etc.

On top of this domain-agnostic engine a geo semantics layer adds handling of space and time coordinates on regular and irregular grids. A rasdaman server offers its geo datacube functionality via the OGC datacube standards, in particular the OGC Web Coverage Processing Service (WCPS) datacube analytics language. Internally, WCPS queries are translated into SQL/MDA and executed in the multi-parallel, distributed evaluation engine.

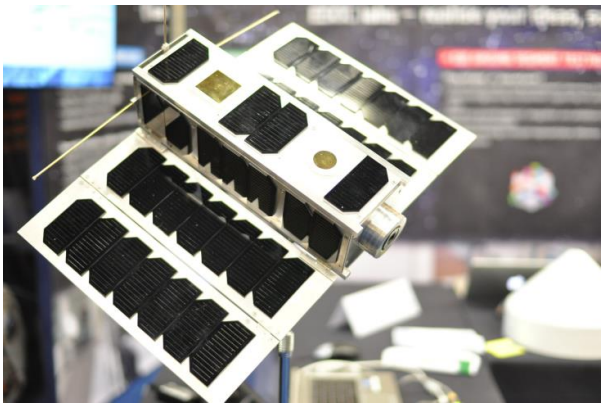

Figure 1: ESA OPS-SAT (source: [10])

Figure 2: OPS-SAT camera capturing frosty fjord (source: [8])

The rasdaman technology is in operational, industrial use since many years forming, among others, the backbone of the first location-transparent datacube federation, EarthServer [2][8].

3.3 Challenges Encountered

Deploying rasdaman on OPS-SAT raised several challenges: a new architecture (ARM Cortex A9 as opposed to Intel-based architecture); constrained environment in terms of computing resources on board (only 1 GB of RAM, less than 16 GB disk space); a slow uplink bandwidth which limited the size of packages that could be uploaded on board in reasonable time; and more. Below we discuss some of the core issues encountered.

Compiling rasdaman for the precise CPU architecture of the satellite proved difficult without access to the actual board. Efforts to do it on a BeagleBone Black board were not successful due to compatibility issues and instabilities. In the end the only solution that worked was cross-compiling in a QEMU virtual machine with the same target processor architecture and 2 GB RAM.

To accommodate for the minimal environment it was necessary to strip away many features that otherwise are a fairly essential part of rasdaman: the OGC Web services frontend that deploys on top of rasdaman as a Java Web Application, support for importing and exporting data in many formats via the GDAL library, as well as special support for NetCDF and GRIB data format, etc. In the end the result was a 10 MB package containing only the core rasdaman engine (written in C++), with support for import / export of data in PNG, JPEG, and BMP formats.

In orbit the OPS-SAT GPS was malfunctioning so it was not possible to demonstrate geo-referencing; however, we consider this not a relevant drawback as the main point of the experiment is the on-board processing of dynamic queries on images acquired; for map production on a commercial-grade satellite GPS would be available with no doubt.

Further, the OPS-SAT downlink did not support TCP/IP initially; surprisingly it became available experimentally about a week before our experiment was scheduled for running on board. Therefore, the workaround implemented earlier had to be used. In future, communication is expected to be more straightforward with some standard TCP/IP stack.

Altogether, a large part of the issues encountered were due to (i) the highly experimental setup and (ii) a general tendency in the on-board IT infrastructure to be rather specific and by far not compliant with IT common industry standards.

4 EVALUATION

4.1 Experiments

Our experiments aimed at demonstrating the value of running rasdaman on a satellite in orbit. The main hypotheses are:

(1) Edge computing is more efficient - we ship computing queries to the data (very low bandwidth needed), instead of data to the computing (high bandwidth needs and expensive for satellites).

(2) Edge nodes, such as satellites, are power and resource constrained, in contrast to large datacenters. Rasdaman is a database engine for multidimensional array data, such as satellite images, written in C++ and heavily optimized for minimal resource usage. Therefore, a datacube engine like rasdaman fits well for deployment on a tiny computer aboard a satellite.

(3) As a database engine, rasdaman allows continuously acquiring and storing large amounts of imagery. This enables much more autonomous operation, as well as flexibility for running ad-hoc queries as needed during the infrequent and short windows for communication with a ground station. Effectively, a satellite becomes a dynamic and agile edge computing node, in contrast to the traditional pre-programmed mode of operation.

(4) At the same time, rasdaman is capable of scaling to make efficient use of powerful computing resources on cloud or supercomputer infrastructure. Through federation it is possible to connect services running on satellites with services running on ground, for a seamless experience when performing data science tasks.

To illustrate (1) we devised several queries on the imagery inserted into rasdaman. As is typical, they reduce the original size, e.g. aggregating results to one or (with a timeseries) a list of scalars; scaling down for quick visual inspection on ground; spatial or channel subsetting; threshold or classification queries resulting in binary images; encoding to lossy JPEG.

The queries that demonstrate (2) perform heavier image processing and analysis, such as edge detection, histogram calculation, timeseries aggregation. The goal is to show that rasdaman is capable of answering queries that take longer, or a lot of data to evaluate.

Query (3) involves continuously acquiring and storing images in a 2D mosaic placed in a 3D timeseries datacube. This should work in a rolling archive fashion to accommodate for the limited disk space so that older data is removed from the datacube when the

disk is filled up. Space usage would be further reduced through support for filtering image acquisition only over areas of interest given as polygon boundaries.

Finally, federating rasdaman nodes (point 4) requires TCP/IP support for communication between the federated machines. OPS-SAT did not have TCP/IP which prevented (4) from being evaluated. An experimental TCP/IP capability got added to OPS-SAT late in the project, about a week before our experiment was scheduled for running on board. Therefore, the proprietary stack had to be kept.

The experiment steps undertaken are presented below. The first five steps were implemented as actions in our NMF app, executed from ground through the Consumer Test Tool (CTT):

1. A "Start rasdaman" command executes the startup routines of the rasdaman engine.
2. A daily collection (table in SQL-speak) is created named *Images_YYYYMMDD* holding the set of 2D images acquired today. This is done by executing the rasql query

   create collection Images_20211027 RGBSet
3. Similar to the previous action, an "Initialize timeseries" action creates a collection for building a 3D timeseries datacube of RGB pixels, and initializes it with a 3D array of one dummy point (updated later on in step 5):

   create collection C RGBSet3

   insert into ImageTimeseries
   values marray m in [0:0,0:0,0:0]
   values {0c,0c,0c}
4. Take a PNG picture with the camera and insert into the daily collection with a query as below:

   insert into Images_20211027
   values decode($1, "png")

   where the *$1* variable is a placeholder for the image acquired, such as *snaps/20211022_073847_myPicture.png*. The full command invocation (which we generally suppress in this paper for brevity) is:

   rasql -q "insert into Images_20211027
   values decode($1, \"png\")"
   -f snaps/20211022_073847_myPicture.png
5. Take a PNG picture with the camera and extend the 3D timeseries with this timeslice:

   update ImageTimeseries as c
   *set c[sdom(c)[0].hi + 1, *:*, *:*]*
   assign decode($1, "png")

   where the *$1* variable again gets replaced by the is a placeholder for an image required, such as *snaps/20211022_073847_myPicture.png* .

   This action is repeated several times one after another to build several slices into the timeseries. Due to the lack of working GPS on board, the datacube is not a real timeseries of imagery sequentially taken over the same spot on Earth. That said, for the purpose of our experiments we can disregard the image contents as the goal was to demonstrate queries over 3D timeseries, which was successful.

The next set of actions contains rasql queries executed via SpaceShell on the imagery acquired previously, rather than being hard-coded into the NMF application. This allows us to modify them on-the-fly if necessary, in response to the results we get. The rasql commands are executed in the shell via the rasql command-line utility, therefore whenever shell commands appear below then quotes are escaped and further necessary rasql parameters are present which have been omitted before. Further, the code has been reformatted to fit the paper layout. Table 1 gives a synopsis of averaged runtimes observed.

6. Download downscaled images extracted from the timeseries for inspection:

   for i in {1..5}; do
   rasql -q
   "*select encode(scale(c[$i,*:*,*:*], 0.1), \"jpeg\")*
   from ImageTimeseries as c"
   done

   The shell loop executes the command five times to allow averaging of results for better accuracy. This loop is performed on all subsequent queries, too, but left out for an easier read of the code.
7. Download a full-size image from the 2D collection (we list only the query as such and omit the shell commands around it):

   select encode(c,"jpeg")
   from Images_20211027 as c
8. Download a random slice from the 3D timeseries in original resolution (likewise omitting the shell commands):

   select encode(c[1,:*,*:*],"jpeg")*
   from ImageTimeseries as c
9. Rudimentary cloud cover assessment on the slices of the 3D timeseries, which is saved as a comma-separated values (CSV) file of cloud cover percentages for each slice:

```
select encode(
  marray i in [1:sdom(c)[0].hi]
  values ( (float) count_cells(
                    c[i[0],*:*,*:*].red   > 250
                and c[i[0],*:*,*:*].green > 250
                and c[i[0],*:*,*:*].blue  > 250
                  )
               / 3981312f ),
   "csv" )
from ImageTimeseries as c
```

Sample output might look like this:
12.53,99.312,34.88

10. Average of the values of each slice, similarly saved as a CSV file of one value per slice:

```
select encode(
  marray i in [1:sdom(c)[0].hi]
  values avg_cells(c[i[0],*:*,*:*],
  "csv")
from ImageTimeseries as c
```

11. Histogram of each channel, saved as a CSV file; an extra shell loop iterates over the RGB bands:

```
for band in red green blue;
do {
  rasql –q
 "select
    encode(
      marray i in [0:255]
      values count_cells(
                c[1:300,1:300].${band} = i[0] ),
      \"csv\" )
  from Images_20211027 as c" \
  --user XXXX --passwd XXXX
  --out file --outfile histogram_${band};
} 2> histogram_${band}_time.txt;
done
```

12. Apply white-balance correction and contrast stretching on each image *$i*: in the 3D cube:

```
select
   encode(
     (char)
     (
       ((c[$i,*:*,*:*]*{1.8, 1, 1}) - 30.0) /
215.0) * 255
     ),
    "jpeg" )
from ImageTimeseries as c
```

On a side note, this slicing is only done for the purpose of extracting and downloading 2D images (see later); on the whole datacube this correction and stretching could be done in a single step (note that the delivery format is changed to NetCDF to accommodate 3D):

```
select
 encode( ( c*{1.8, 1, 1} - 30.0) / 215.0 * 255,
"netcdf" )
from ImageTimeseries as c
```

13. Edge detection with a Sobel kernel over a 2D image (the Sobel kernel is indicated verbatim while alternatively it could be conveniently stored as an object in the database itself, which would simplify the query):

```
select
  avg_cells(
    sqrt(
      pow(
        marray p1 in [1:100,1:100]
        values
          ( condense +
            over k1 in [-1:1,-1:1]
            using ( <[-1:1,-1:1]
                      1,0,-1; 2,0,-2; 1,0,-1>
                       [k1[0], k1[1]]*c[p1[0]
                  + k1[0], p1[1] + k1[1]].red
                  )
          ),
          2.0
      )
      +
      pow(
        marray p2 in [1:100,1:100]
        values
          ( condense +
            over k2 in [-1:1,-1:1]
            using ( <[-1:1,-1:1]
                      1,0,-1; 2,0,-2; 1,0,-1>
                       [k2[0], k2[1]]*c[p2[0]
                  + k2[0], p2[1] + k2[1]].red
                  )
          ),
          2.0
      )
    )
  )
from Images_20211027 as c
```

14. Derive a cloud mask from each slice $i in the timeseries (the loop again is omitted):

```
select
  encode(
    (char)
    (
      ( c[$i,*:*,*:*].red >200 and
        c[$i,*:*,*:*].green > 220 and
        c[$i,*:*,*:*].blue > 220
      ) * {255c,0c,0c}
    ),
    "png" )
from ImageTimeseries as c
```

4.2 Results

The above experiments have been conducted in-orbit in December 2022. We discuss the extraction and download cases (queries 6 through 14 above) in turn.

Query 6: Five timeseries slices downscaled to 10% of the original size are shown in Figure 3. Extracting and saving each image on the on-board disk took 1.77, 2.07, 2.03, 1.73, and 1.78 seconds.

Query 9: Calculating cloud cover percentage for each slice was performed in 28.31 seconds. The output was delivered in CSV format as it consists of a single number. Results delivered were: 2.361031e-05%, 0%, 9.544592e-06%, 0%, and 0%.

Query 10: The query that calculates average values for each band of the images took 14.9 seconds to execute. The resulting CSV data are visualized as a stacked bar chart in Figure 4 and 5. Images with higher averages are brighter, and vice versa. Scaling the band averages for each image to 100%, corresponding to an intensity of 255, reveals that the on-board camera has a bias on the blue and green channels. Indeed, all images show a significant bluish tint (Figure 4).

Query 11: Instead of an average we can obtain the detail distribution in the form of channel histograms. Executing the histogram query for the red, green, and blue band took 11.48, 8.25, and 8.11 seconds respectively. Figure 7 shows a chart of the query result.

Query 7 and 8: Based on the small image previews inspected we decided to download four image slices in original resolution. The times for extracting each and saving on disk were 4.98, 1.86, 1.82, and 1.80 seconds. The first full slice is shown in Figure 6 (scaled down to paper column width).

Query 12: This query was executed in an attempt to correct the sensor color bias and improve the brightness and contrast of the images. It was executed for the first four images of the timeseries in 16.17, 15.84, 28.35, and 23.27 seconds. The results are shown on Figure 7 and 8, a significant improvement to the original imagery as acquired by the camera as shown in Figure 3.

Query 13: The edge detection query took 17.37 seconds to evaluate on board, which is understandable as it is a computationally expensive operation and only one single, slow CPU core was available (which additionally is shared with the operating system and potentially other processes).

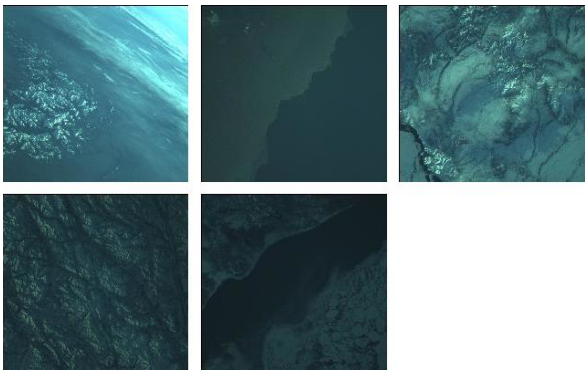

Figure 3: Downscaled images for quick inspection on ground

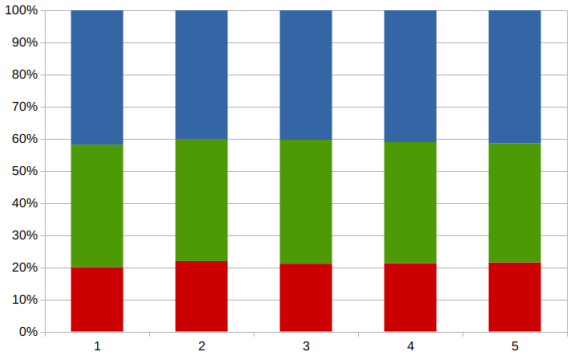

Figure 4: Average value of the images bands (Y axis) for each of the 5 images (X axis)

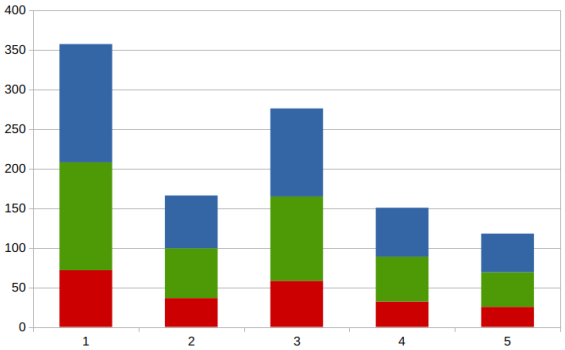

Figure 5: Band averages scaled to 100% (Y axis) for each of the 5 images (X axis)

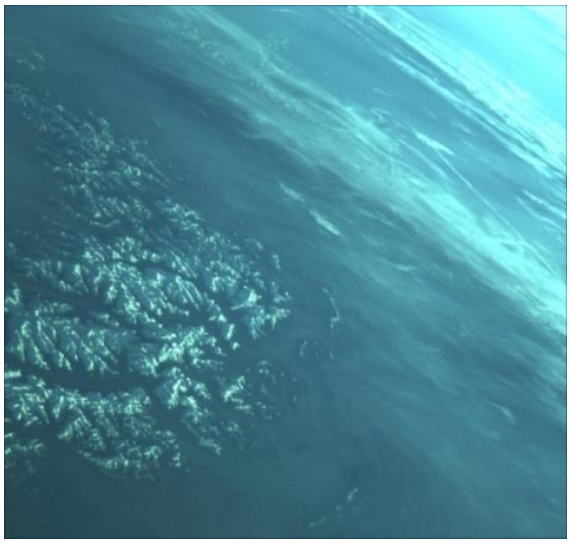

Figure 6: First slice in full resolution (downscaled for printing)

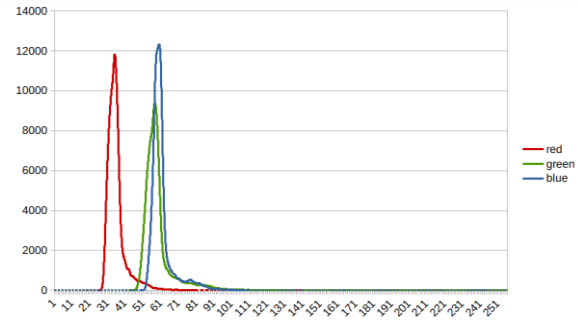

Figure 7: Channel histograms showing the count (Y axis) of each pixel value (X axis) in one image.

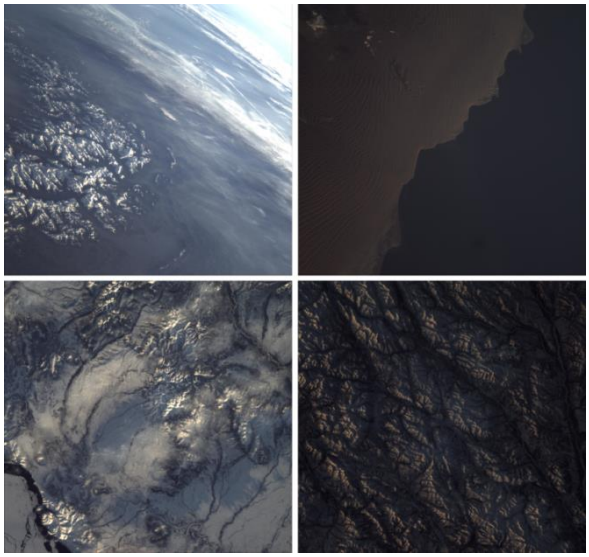

Figure 8: White-balance and contrast correction

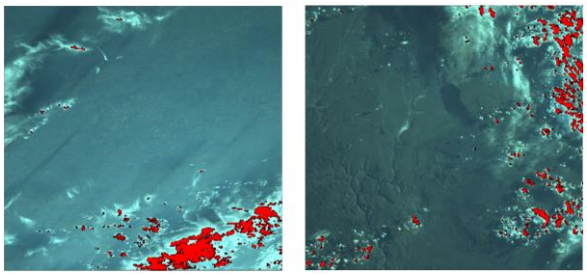

Figure 9: Cloud detection

Query 14: In contrast to getting a cloud-cover percentage, this query derived a cloud mask that highlights the areas covered by clouds in red (Figure 9). Cloud-cover calculation is based on a rudimentary threshold method, and occasionally mistakes snow for clouds as well. As the images are largely cloud-free, the resulting cloud masks are almost completely empty and not shown here. Execution times were 12.4, 11.2, 10.83, and 11.98 seconds for each of the first four slices respectively.

The identical queries, as run on the satellite, were repeated on ground on a standard desktop PC with 2x Intel Xeon with a total of 12 virtual cores running at 1.90 GHz, 64 GB RAM, as well as a 512 GB SSD root partition and a 6TB 7200 RPM SATA data partition. Data downloaded have been imported in the local database to resemble the exact same data situation. In Table 1 below we compare performance results obtained; recall that on-board a single-core ARM processor is in use. Both computers were running rasdaman community 9.

As can be expected, the more powerful hardware improves performance significantly. Obviously, having hardware on board that is only comparable to common office standards can lead to interactive real-time query processing on datacubes in space. We expect significant gains once hardware and software components in space start following common IT industry standards.

Table 1: Query runtimes (secs, averaged)

query#	**6**	**7**	**8**	**9**	**10**
satellite	2.34	2.61	2.61	28.31	14.9
desktop	0.1	0.32	0.32	3.45	1.43
query#	**11**	**12**	**13**	**14**	
satellite	9.28	20.90	17.37	11.6	
desktop	1.05	1.79	0.94	1.48	

5 CONCLUSION

We have presented the ORBiDANSe project which is aiming to bridge the gap between two technologies hitherto far apart from each other:

- user-oriented datacube technology easing access, analytics, and fusion of massive spatio-temporal Earth data;
- nano-satellites which, in the spirit of IoT, can be considered edge devices with specific characteristics, such as a high data delivery rate stretching downlink bandwidth to its limits.

Innovative contributions made are in particular:

- A datacube engine usually scaled up to run in supercomputing centers has been scaled down to run in an extremely limited hardware setup;
- Controlling satellite image acquisition, processing, and data download could be accomplished through common APIs;
- Query interfaces are based on adopted standards, so no special tools are needed for gaining insight from the satellite observations. In fact, any standards compliant client ought to be able to connect.

As we have learnt, on-board processing environments are far away from IT standards common elsewhere. Lots of workarounds had to be found, and interaction was not as seamless and user-friendly as one is used to on Earth. We hope, therefore, to make a case for satellite payload operators to move towards standard components and interfaces that are easy to use for common developers relying on industry-established, stable solutions on board, too. NASA SpaceCube appears a step in the right direction.

We frequently hear from space experts (and agencies) that all pixels invariably must be brought to ground and archived as no data should get lost. While we do not object to this position on principle we contend that today there is already a good basic supply of complete spatial and temporal coverage; we see satellites with on-board datacube query processing as a complementary service adding fast and flexible ad-hoc insight to the basic supply. We believe, therefore, that in future datacube services will contribute an important facet towards "any insight, any time". Ultimately, such an approach has a potential for democratizing satellite data access as common tools – in case of rasdaman, ranging from OpenLayers over NASA WorldWind and QGIS to python and R – become a means to talk to satellites directly, without the need to wait for data becoming published through the ground segment. Obviously, proper access control to the satellite needs to be established in parallel.

As a next step we plan operational service deployments, first in single-instance scenarios and subsequently federated. In such a federated scenario, a client may submit some complex decision support query to a data center; the rasdaman instance there finds out that data are missing and spawns a sub-request to the cubesat, and merges its locally computed results with the cubesat response into the final result sent back to the user. As rasdaman is already cloud-parallelized, queries can be distributed automatically between ground and space instances. Technically, this fog computing scenario ties the satellites into the mashup as edge devices; particularly interesting will be to observe – and if necessary improve – distributed query processing optimization in presence of highly asymmetric processing and bandwidth capabilities within such a federation.

ACKNOWLEDGMENT

This work has been supported by the German Ministry of Economics and Energy under grant ORBiDANSe.

REFERENCES

[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, P. Mazzetti, J. Ungar, R. Barbera, D. Barboni, A. Beccati, L. Bigagli, E. Boldrini, R. Bruno, A. Calanducci, P. Campalani, O. Clement, A. Dumitru, M. Grant, P. Herzig, G. Kakaletris, J. Laxton, P. Koltsida, K. Lipskoch, A.R. Mahdiraji, S. Mantovani, V. Merticariu, A. Messina, D. Misev, S. Natali, S. Nativi, J. Oosthoek, J. Passmore, M. Pappalardo, A.P. Rossi, F. Rundo, M. Sen, V. Sorbera, D. Sullivan, M. Torrisi, L. Trovato, M.G. Veratelli, S. Wagner, "Big Data Analytics for Earth Sciences: the EarthServer Approach", International Journal of Digital Earth 0(0)2015.

[3] P. Baumann, "OGC Web Coverage Processing Service (WCPS) Language Interface Standard", OGC document 08-068r3, https://www.ogc.org/standards/wcps, seen 2022-07-27.

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

[5] P. Baumann, "Language Support for Raster Image Manipulation in Databases", Int. Workshop on

Graphics Modeling, Visualization in Science & Technology, Darmstadt, Germany 1992, pp. 236 – 245, DOI 10.1007/978-3-642-77811-7_19 .

[6] P. Baumann, E. Hirschorn, and J. Masó, "OGC Coverage Implementation Schema with Corrigendum", OGC 09-146r8, 2019-10-28.

[7] H. Chen, L. Li, Z. Zhong and J. Li, "Approach for earth observation satellite real-time and playback data transmission scheduling", in Journal of Systems Engineering and Electronics, 26(5)2015, pp. 982-992, DOI 10.1109/JSEE.2015.00107.

[8] EarthServer, "EarthServer Datacube Federation", https://earthserver.eu, seen 2022-07-27.

[9] ESA, "First OPS-SAT photos capture frosty fjord", https://www.esa.int/ESA_Multimedia/Images/2020/08/First_OPS-SAT_photos_capture_frosty_fjord, seen 2022-07-27

[10] ESA, "OPS-SAT: your flying laboratory", https://www.esa.int/ops-sat, seen 2022-07-27.

[11] D. Giggenbach, J. Horwath, and M. Knapek, "Optical data downlinks from Earth observation platforms", SPIE 7199, Free-Space Laser Communication Technologies XXI, 719903, February 2009, DOI 10.1117/12.811152.

[12] G. Giuffrida et al., "The Φ-Sat-1 Mission: The First On-Board Deep Neural Network Demonstrator for Satellite Earth Observation", IEEE Transactions on Geoscience and Remote Sensing, vol. 60, 2022, pp. 1-14, DOI 10.1109/TGRS.2021.3125567.

[13] ISO, "9075-15:2019 SQL/MDA (Multi-Dimensional Arrays)", https://www.iso.org/standard/67382.html, seen 2022-07-27.

[14] L. Jun, H. Chen, and J. Ning, "A Data Transmission Scheduling Algorithm for Rapid-Response Earth-Observing Operations", Chinese Journal of Aeronautics 27 (2014): 349-364.

[15] G. Labrèche, D. Evans, D. Marszk, T. Mladenov, V. Shiradhonkar, and V. Zelenevskiy, "Agile Development and Rapid Prototyping in a Flying Mission with Open Source Software Reuse On-Board the OPS-SAT Spacecraft", AIAA 2022-0648, AIAA SCITECH Forum, January 2022.

[16] NASA, "SpaceCube: a Family of Reconfigurable Hybrid On-Board Science Data Processors", https://spacecube.nasa.gov, seen 2022-07-27 .

[17] OGC, "OGC Web Coverage Service (WCS) 2.1 Interface Standard – Core". OGC document 17-089r1, http://docs.opengeospatial.org/is/17-089r1/17-089r1.html, seen 2022-07-27.

[18] rasdaman, "WCPS: Power and Speed Demo", https://standards.rasdaman.com/demo_power.html, seen 2022-07-27 .

[19] M. Sudmanns et al. "Big Earth Data: Disruptive Changes in Earth Observation Data Management and Analysis?", International Journal of Digital Earth, Vol. 13,7 832-850. 14 March 2019, DOI10.1080/17538947.2019.1585976.

[20] USGS, "U.S. Landsat Analysis Ready Data", USGS Report 2018-3053, Index ID fs20183053, September 2018, DOI 10.3133/fs20183053, seen 2022-07-27.

[21] Y. Wang et al. "Satellite Edge Computing for the Internet of Things in Aerospace", Sensors vol. 19, 20, 4375. October 2019, DOI 10.3390/s19204375.

AUTHOR BIOGRAPHIES

Dimitar MISEV is a Postdoc researcher in Computer Science at Jacobs University Bremen, and Director of Product Development at rasdaman GmbH. His research focuses on the management and processing of large multidimensional array data within database systems, building large-scale datacube services holding Petabytes of data, and contributing to ISO SQL standardization efforts as main author of the SQL datacube extension.

Peter BAUMANN is Professor of Computer Science and entrepreneur. At Jacobs University he researches on flexible, scalable datacube services and their application in science and engineering. With the rasdaman engine he and his team have pioneered datacubes and Array Databases, and have set the de-facto standard for datacube services, documented by 160+ scientific publications, international patents and numerous high-ranking innovation awards. As founder and CEO he leads the successful international commercialization of rasdaman. Since many years, Baumann is leading datacube standardization in ISO, OGC, and EU INSPIRE. Baumann is chair, IEEE GRSS Earth Science Informatics Technical Committee; co-chair, OGC Coverages.SWG and Coverages.DWG and BigData.DWG; editor, ISO 19123 suite.