Open and scalable analytics of large Earth observation datasets: From scenes to multidimensional arrays using SciDB and GDAL

Marius Appel a,*, Florian Lahn a, Wouter Buytaert b, Edzer Pebesma a

a Institute for Geoinformatics, University of Münster, Heisenbergstr. 2, 48149 Münster, Germany
b Department of Civil and Environmental Engineering, Imperial College London, Skempton Building, SW7 2AZ London, United Kingdom

A R T I C L E   I N F O
Article history:
Received 1 February 2017
Received in revised form 19 January 2018
Accepted 19 January 2018

Keywords:
Earth observation
Database systems
Scalable analytics
Open science
Big data
Spatial information science

A B S T R A C T
Earth observation (EO) datasets are commonly provided as collection of scenes, where individual scenes represent a temporal snapshot and cover a particular region on the Earth’s surface. Using these data in complex spatiotemporal modeling becomes difficult as soon as data volumes exceed a certain capacity or analyses include many scenes, which may spatially overlap and may have been recorded at different dates. In order to facilitate analytics on large EO datasets, we combine and extend the geospatial data abstraction library (GDAL) and the array-based data management and analytics system SciDB. We present an approach to automatically convert collections of scenes to multidimensional arrays and use SciDB to scale computationally intensive analytics. We evaluate the approach in three study cases on national scale land use change monitoring with Landsat imagery, global empirical orthogonal function analyses of daily precipitation, and combining historical climate model projections with satellite-based observations. Results indicate that the approach can be used to represent various EO datasets and that analyses in SciDB scale well with available computational resources. To simplify analyses of higher-dimensional datasets as from climate model output, however, a generalization of the GDAL data model might be needed. All parts of this work have been implemented as open-source software and we discuss how this may facilitate open and reproducible EO analyses.

© 2018 The Authors. Published by Elsevier B.V. on behalf of International Society for Photogrammetry and Remote Sensing, Inc. (ISPRS). This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Today’s abundant availability of Earth observation (EO) datasets enables us to improve our understanding of the environment on regional, continental, and planetary scales. Many space agencies have opened up their archives and provide EO datasets for free. This includes data from long running missions like MODIS and Landsat, but also from recently launched satellites like the Sentinels. Although this is of great value for observing and understanding processes on Earth, actually using these datasets in complex analyses can be challenging for two major reasons. First, data size stresses existing analytical systems and software. Datasets easily exceed memory and storage capacities of single computers (Lewis et al., 2017) and, at the same time, computation times of complex algorithms may become infeasible if they do not scale with available hardware resources (Wagner, 2015). Second, the organization of EO datasets as collection of scenes can be inefficient. For example, accessing and analyzing time series of high resolution Landsat imagery may yield nonoptimal data access patterns, where values of a single time-series come from many files. As a result, scientists must spend an increasing amount of time on data management and parallelization of algorithms instead of developing, improving, and validating models.

In this paper, we approach this problem by using and extending the data management and analytics system SciDB (Stonebraker et al., 2013), representing EO datasets as multidimensional arrays. The aim is to present a workflow transferring scenes to multidimensional arrays on the one hand and to evaluate the scalability of relatively complex statistical EO analyses on the other hand. For the former, we discuss and demonstrate SciDB extensions (Appel et al., 2016) which simplify the ingestion of file-based scenes using the Geospatial Data Abstraction Library (GDAL) (Warmerdam, 2008) and enable in-database spatiotemporal mosaicing and joins. The scalability of the resulting system is evaluated in three study cases including the computation of empirical orthogonal functions, land use change monitoring, and comparing climate model output to satellite-based observations.

The paper is organized as follows: Section 2 gives an overview of alternative approaches to the problem we address. Section 3
introduces how the multidimensional array-based data analysis and management system SciDB can be used to store and process EO datasets before our extensions to interface with scene-based EO data are discussed. Section 4 presents developed study cases and their results before Section 5 discusses the approach and its limitations and Section 6 concludes the paper. Detailed information on the experimental setup is provided in Appendix A. Supplementary material to the paper with full resolution results of the study cases has been published at Zenodo and code for a demonstration how to run the study cases on small datasets is available at GitHub (see Appendix B).

2. Related work

Google Earth Engine (GEE) (Gorelick et al., 2017) is a cloud platform that provides access to most freely available EO datasets, as well as the ability to compute with these data. GEE offers a web-based platform where EO datasets can be processed using JavaScript or Python. Examples of analytics derived with GEE include the global forest change inventory for 2000–2014 based on time series of Landsat imagery (Hansen et al., 2013). Although the GEE is scientist friendly in the sense that there are no financial costs to its use, and large computational capacity can be obtained with relative ease, some have called it “evil for science” (Inglada, 2016) because there is no access to the details of the underlying algorithms (closed source) and a business model or service level agreement is missing, meaning that the service might be discontinued at any time. In any case, the underlying system is not extensible by users, except from the JavaScript interface.

From a technological perspective, there is a vast amount of open-source solutions available for big data storage and processing. Although their development mostly focused on processing tabular data and not multidimensional scientific arrays (Stonebraker et al., 2013), many of those including frameworks such as Hadoop with its distributed file system HDFS (Shvachko et al., 2010) and the MapReduce processing model (Dean and Ghemawat, 2008), or Apache Spark (Zaharia et al., 2016) in principle can be used for Earth observation analytics. A framework that has been specifically developed for scalable raster data processing is GeoTrellis (Kini and Emanuele, 2014). GeoTrellis is based on Spark and its use for EO analysis platforms has been shown e.g. in Goor and Dries (2017). The underlying architecture is still file-based, meaning that it is not trivial to combine data from e.g. overlapping scenes in analyses besides running algorithms in parallel over many tiles.

An alternative approach is to represent EO datasets as multidimensional arrays and to use array databases for storage and analyses. Within the EO community, Rasdaman (Baumann et al., 1998) and SciDB (Stonebraker et al., 2013) are popular array databases. Rasdaman has been successfully used for managing data in a variety of applications within the EarthServer project (Baumann et al., 2016) but besides image processing, demonstrated analyses have been computationally rather simple and do not go beyond selection, aggregation, resampling, or calculation of simple statistics. To allow for more complex analytics, key mathematical operations like linear algebra routines are missing and would require external processing. Furthermore, at the time of writing this paper, the open-source community edition of Rasdaman neither scales individual computations to multiple processors nor does it distribute memory consumption to multiple nodes.

Planthaber et al. (2012) use the open-source edition of SciDB to store and process level-1B MODIS imagery. Their results are promising in terms of scalability but presented tools are not applicable to imagery from other satellites. Furthermore, the demonstrated RGB image generation and vegetation index computation are still relatively simple. The scalability of SciDB in more complex analyses including regression, clustering, and statistical testing has been evaluated by Taft et al. (2014). They compared different data management approaches to analyse genomics data including relational and column-store databases, Hadoop, and SciDB. Lu et al. (2016) use SciDB to scale land use change detection on moderately sized MODIS imagery. We build on their finding that SciDB can scale change detection algorithms on EO image time-series in one of the study cases. Compared to Lu et al. (2016), this paper presents an approach for constructing multidimensional arrays from imagery in any format readable by GDAL and also demonstrates study cases that are more difficult to scale. Tan et al. (2017) use some of the software contributions described in this paper and simulate forest fire spread. Their analysis, however, does not scale because only the preprocessing (deriving slope from a digital elevation model) runs in the database whereas the following simulation steps are processed externally.

The climate science community almost exclusively organizes their data as multidimensional NetCDF files (Rew and Davis, 1990). The data model itself is array-based but does not scale on its own and still needs efficient implementations of basic algorithms. The Python package xarray (Hoyer and Hamman, 2017) provides such an implementation by mimicking functionality from standard tabular data structures (pandas) for multidimensional arrays with the same data model as the NetCDF file format. Its integration of the Dask package for parallel computing (Rocklin, 2015) furthermore allows for scalable processing even in distributed environments. Xarray strongly simplifies how NetCDF files in the Geosciences can be processed but does not address difficulties with the organization of EO imagery as scenes. The Open Data Cube (ODC) Initiative1 and related national data cube implementations (see Lewis et al., 2017, for example) provide tools to build, access, and process data-cubes (three-dimensional spacetime arrays) for a large number of EO products from Python and thereby also interface xarray.

A detailed technical evaluation of different data management approaches including the use of NetCDF files, Rasdaman, and SciDB has been recently conducted by Bakcsa et al. (2016). They conclude that multidimensional array-based databases show an enormous potential especially for analyses of higher-level EO products, but also point out that data loading is not trivial and a significant drawback for practical applications. Mehta et al. (2017) support this statement when they compare SciDB to other Big data systems including Dask and Spark for image analysis workloads in other disciplines. Haynes et al. (2017) furthermore demonstrate that SciDB outperforms traditional relational databases for zonal raster analyses.

3. Earth observation datasets as multidimensional arrays

A multidimensional array $A$ can be seen as a function that maps coordinates from an $n$-dimensional index space to an $m$-dimensional attribute space $A : D \rightarrow V$ with $D \subset D_1 \times \ldots \times D_n$ and $V \subset V_1 \times \ldots \times V_m$, where all $D_i$ are finite and totally ordered (see Baumann, 1999; Baumann and Holsten, 2011, for equivalent definitions). We denote $D_i$ as the $i$-th dimension, a specific element $p \in D$ a cell, $V_i$ as the $i$-th attribute, and the function value $A(p)$ as the cell value with regard to $p$. For simplicity, dimensions are usually represented as integer indexes through the finiteness property allows for defining an invertible transformation (bijection) from integers to arbitrary dimensions (e.g. latitude). If a dimension is regular, i.e. consecutive elements of $D_i$ have a constant separation distance, this reduces to a simple affine transformation by an offset and cell size or interval specification. To map multidimensional

---

arrays to computer memory, dimensions are flattened to a single artificial integer dimension, which can be translated to memory addresses. Examples for flattening are typical row-major or column-major order in programming languages. Dimensions are a natural index for the data and looking up cell values at a given position is efficient because memory addresses can usually be computed in $O(1)$.

The array model fits very naturally to EO data: a single (level 1 and higher) remote sensing image is usually interpreted as an array with two $(row \times column \rightarrow band_1 \times band_2 \times \ldots)$ or three $(row \times column \times band \rightarrow reflectance)$ dimensions. An affine transformation converts row and column integer indexes to spatially referenced coordinates and vice versa. Whether or not array cells or pixels have the same physical size/area depends on the underlying reference system. A time series of images similarly can be seen as a three- $(row \times column \times day \rightarrow band_1 \times band_2 \times \ldots)$ or four-dimensional $(row \times column \times day \times model \rightarrow var1 \times var2 \ldots)$. The climate science community thereby has set up enormous infrastructures to organize their datasets in a standardized way (Taylor et al., 2012; Eyring et al., 2016) for the long term (Weigel et al., 2015).

### 3.1. Scalable analytics with SciDB

The data management and analytical system SciDB (Stonebraker et al., 2013) implements the array model and pays special attention to scalable processing. Arrays are partitioned into equally sized chunks. Chunks are distributed over database instances, which may run on different physical machines in a shared-nothing cluster computing environment. Storage consumption and computational load can be balanced and scale with available resources while chunking helps to preserve data locality. The size of chunks may or may not be equal along the dimensions, and can be specified per array. SciDB’s query language AFL provides built-in functionality to perform arithmetic operations on cell values, subsetting arrays by criteria on dimensions or attributes, aggregation and downscaling over dimensions, reshaping arrays, and array joins. Additionally, SciDB includes scalable implementations of linear algebra routines by interfacing ScaLAPACK (Choi et al., 1992). The underlying storage supports sparsity of arrays such that empty array cells do not consume substantial amounts of memory. Increasing the resolution of EO arrays and introducing empty cells where no observations are available often facilitates dealing with irregular dimensions. For example, it can be convenient to represent time series of Landsat scenes from different swaths as a three-dimensional spacetime array with daily temporal resolution: although the satellite’s revisit interval is 16 days, scenes from adjacent swaths might be taken at dates in between and a daily temporal resolution would handle multiple observations in overlapping areas appropriately. Section 4.2 discusses the benefit of sparse storage in a practical application. Array dimensions in SciDB use 64-bits integers. The Earth’s full circumference in principle can be discretized up to picometers $(12742 \times \pi \times 1000 \times 2^{-64})$.

Though SciDB’s data model, sparse storage, and functionality are well suited to manage EO datasets, Bakcsa et al. (2016) point out that making use of these advantages is not trivial. Loading data from common file formats is not supported and representing space and time as integer array dimensions requires manual bookkeeping of important metadata such as spatial and temporal reference.

We address these limitations by two software extensions: the plugin sci4geo runs within SciDB and adds basic EO-specific functionality while the sci4gdal extension to the geospatial data abstraction (GDAL) simplifies interfacing arrays with file-based scenes. Fig. 1 illustrates how both extensions can be integrated to a scalable SciDB system.

### 3.2. Spacetime arrays in SciDB: the sci4geo plugin

To represent large EO datasets as multidimensional arrays in SciDB, we extended SciDB to (i) define how arrays refer to space and time, (ii) run spatiotemporal overlays, joins, and mosaicing, and (iii) maintain additional EO-specific metadata. The extension adds new operators to SciDB’s query language AFL (Table 1).

#### 3.2.1. Arrays in space and time

To define how arrays refer to space, we assume that horizontal space is represented by two array dimensions. Arrays may still have more dimensions like spectral wavelength, vertical elevation, or sensor identifiers. As in the GDAL data model (Warmerdam, 2008), integer indexes in the two spatial dimensions are transformed to referenced coordinates by a six parameter affine transformation

$$\varphi: \mathbb{Z}^2 \rightarrow \mathbb{R}^2, (x, y) \mapsto (x_0 + a_{11}x + a_{12}y, y_0 + a_{21}x + a_{22}y).$$

Together with additionally stored spatial reference system definitions this allows to physically locate arrays on Earth but assumes imagery being georeferenced, i.e., it is not applicable to raw satellite imagery.

To relate cells of time-series arrays to physical time, we assume time is being represented as a single array dimension. For this
The formula above. In time, \( B \) or cover multiple cells of \( A \) indexes to actual dates (times) with regard to array attributes of overlapping arrays (spacetime join). Therefore, an additional coordinate transformation can be applied to a version from dates (times) to array indexes with regard to array attributes of overlapping arrays (spacetime join). Therefore, our extension does not consider the statistical support of array attributes of overlapping arrays (spacetime join). Therefore, we extended GDAL by a driver for spacetime SciDB arrays that:

- converts GDAL datasets to and from SciDB's binary file format,
- sends and downloads data to and from SciDB via HTTP(s) using the web service shim (Paradigm4, 2016b),
- and maps many two-dimensional GDAL datasets to single multidimensional (spacetime) SciDB arrays using additional GDAL opening or create options.

In the trivial case where analysis is confined to a single scene only, this scene can be represented as a simple two-dimensional SciDB array and ingestion involves a single call of `gdal_translate` as in Fig. 2.

In the general case, however, multiple scenes may cover different spatial regions, have been captured at different dates, or even both. We propose an approach for the conversion of such datasets to a single SciDB array, which calls `gdal_translate` iteratively over all scenes using the same target array name. The first call will create the array while further calls will successively insert scenes at the right position.

If scenes cover different regions, the overall spatial extent of the target array can be specified as additional argument. The resolution and spatial reference system is then taken from the first scene and in each following insertion, the position of the scene in the target array is automatically derived in the database using the overlay operation (Section 3.2.2).

If scenes have been captured at different dates, the particular date or time of a scene must be given as additional create option to each call of `gdal_translate`. During the initial array creation, the temporal resolution (i.e., temporal gap between consecutive array cells) must be furthermore specified. This procedure will then result in a three-dimensional SciDB array with data from many scenes. Fig. 3 illustrates the procedure to generate a SciDB array from multiple scenes.

The procedure above can be combined with other GDAL functionality. If scenes for instance come from different UTM zones,

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eo_arrays()</td>
<td>List geographically referenced arrays</td>
</tr>
<tr>
<td>eo_setsrs()</td>
<td>Set spatial reference of existing arrays</td>
</tr>
<tr>
<td>eo_regnewrs</td>
<td>Register custom spatial reference systems e.g. by importing from <a href="http://spatialreference.org">http://spatialreference.org</a></td>
</tr>
<tr>
<td>eo_extent()</td>
<td>Compute the geographic extent of referenced arrays</td>
</tr>
<tr>
<td>eo_setsrs()</td>
<td>Set the temporal reference of arrays</td>
</tr>
<tr>
<td>eo_gettsrs()</td>
<td>Fetch the temporal reference of arrays</td>
</tr>
<tr>
<td>eo_setmd()</td>
<td>Set key value metadata of arrays and their attributes</td>
</tr>
<tr>
<td>eo_getmd()</td>
<td>Fetch metadata of arrays and array attributes</td>
</tr>
<tr>
<td>eo_over()</td>
<td>Overlay two referenced arrays by space and/ or time</td>
</tr>
<tr>
<td>eo_coords()</td>
<td>Derive physical cell coordinates from indexes</td>
</tr>
</tbody>
</table>

3.3. Creating multidimensional arrays from collections of scenes: the `scidbgdal` driver

EO scenes are provided in a variety of file formats like GeoTIFF, HDF, NetCDF, or even as complex compressed directory structures as for Sentinel 2 imagery (European, 2016). To abstract from particular formats, the Geospatial Data Abstraction Library (GDAL) (Warmerdam, 2008) has been developed. GDAL supports reading and writing a large variety of more than 140 raster file formats and is integrated into practically every Geographic Information System (GIS) and programming language. Among others, it comes with a set of utilities (see Warmerdam, 2016, for a complete list) to convert between file formats (gdal_translate), to warp and reproject images to different coordinate reference systems (gdal_warp), and to combine images in virtual datasets (gdalbuildvrt).

The underlying model of GDAL is essentially two-dimensional with multiple spectral bands (attributes). This model can be limiting e.g. in analyzing image time series or irregular datasets. For example, creating a mosaic from two adjacent overlapping scenes (as from Landsat) would overwrite values of one scene at the overlapping area although the scenes have been captured at different days. Despite this limitation compared to e.g. NetCDF files, GDAL abstracts from file formats and even supports images organized as complex directory structures such as the Sentinel-2 format (European, 2016). Therefore, we extended GDAL by a driver for spacetime SciDB arrays that:

- converts GDAL datasets to and from SciDB's binary file format,
- sends and downloads data to and from SciDB via HTTP(s) using the web service shim (Paradigm4, 2016b),
- and maps many two-dimensional GDAL datasets to single multidimensional (spacetime) SciDB arrays using additional GDAL opening or create options.

3.2.2. Spacetime overlays, mosaicing, and joins

The information how arrays relate to space and time can be used to identify which cells of two arrays spatially or temporally overlap. We refer to this operation as the overlay of two arrays. To overlay two arrays by space, their (inverted) affine transformation can be used to compute array indexes of a first array \( A \) that relate to array cells in a second array \( B \) by:

\[
\text{over}(p) := (\varphi_B^{-1} \circ \varphi_A)(p),
\]

where \( p \) represents an integer array index of array \( A \) and the result \( \text{over}(p) \) equals the index of the array cell in array \( B \). Both arrays are assumed to have the same coordinate reference system. If not, an additional coordinate transformation can be applied to \( \varphi_A(p) \) in the formula above. In time, \( \text{over} \) requires the conversion from array indexes to actual dates (times) with regard to array \( A \) and the conversion from dates (times) to array indexes with regard to array \( B \). \( \text{Over()} \) will not yield integer coordinates if cells of \( A \) are within one or cover multiple cells of \( B \).

We implemented the new SciDB operator \( \text{eo_over} \) that evaluates over for all cells of a first input array and creates attributes for the corresponding indexes at the second input array. The operator supports spatial and temporal overlays and can be used to merge multiple non-overlapping arrays with the same attributes into a single larger array (spacetime mosaicing) and to combine attributes of overlapping arrays (spacetime join). Therefore, \( \text{eo_over} \) must be combined with existing SciDB operators \( \text{insert}, \text{redimension}, \text{and cross_join} \) (see Section 4.1 for a real-world example, where we join climate model output with satellite-based observations by space and time).
one can use gdalwarp to warp or reproject the imagery to a common reference system before. Other utilities such as gdal_merge.py or gdalbuildvrt can be useful to combine files e.g. for different spectral bands in one dataset before running gdal_translate. Similar to the described ingestion procedure, our GDAL driver supports downloading temporal slices and arbitrary spatial and spectral subsets from SciDB arrays.

4. Study cases

Three study cases below demonstrate how the proposed workflow can be applied in order to represent different EO datasets and how relatively complex statistical operations scale. For each study case, we discuss how the data can be represented in and ingested to SciDB and we assess scalability of following analyses by measuring computation times for varying computational resources. All computations have been performed on the same machine with the same setup using 16 SciDB instances. In experiments on scalability, the number of available processors has been varied and median computation times have been derived from five repetitions. Details on the experimental setup can be found in Appendix A.

4.1. EOF analysis of global precipitation measurements

In this study case, we compute spatial empirical orthogonal functions (EOFs) of a large precipitation dataset from the Tropical Rainfall Measuring Mission (TRMM) (Huffman et al., 2007). EOF analysis is a standard exploratory method for dimension reduction and finding typical patterns. It is equivalent to principal component analysis and has been discussed in detail e.g. in Hannachi et al. (2007) and Cressie and Wikle (2015). Besides being an exploratory tool, EOFs of precipitation data can be used in subsequent temporal downscaling (Mahmud et al., 2015).

Using SciDB and GDAL extensions as described in Sections 3.2 and 3.3, we ingest scenes of the daily merged TRMM product (NASA, 2015) from 1 January 1998 to 31 December 2014, that is 6210 images, as a three-dimensional array comprising around 3.5 x 10^8 cells. Each scene of daily precipitation originally comes as a NetCDF file readable by GDAL. We iteratively call gdal_translate (Section 3.3) where each call adds a new temporal slice to the resulting array. Image dates have been extracted from filenames and added as create options to individual gdal_translate calls. The resulting array of size (400 x 1440 x 6210) is then used in the computation of global precipitation EOFs by running the following steps:

1. Remove 317 scenes with missing values.
2. Reshape the three-dimensional array to a dense two-dimensional data matrix X where 576000 columns correspond to spatial pixels and 5893 rows represent recording days.
3. Detrend the time series of individual pixels by computing and subtracting temporal means and store the result as X'.
4. Perform a singular value decomposition (SVD) on the resulting array \( \mathbf{X} = \mathbf{U} \mathbf{S} \mathbf{V}^T \).
5. Reshape the first \( i \) columns of \( \mathbf{V}^T \) (EOFs) to a three-dimensional array with dimensions latitude, longitude, and EOF number.

Fig. 4 shows the resulting first 10 EOFs of the complete dataset. Most EOFs show a clear pattern of the intertropical convergence zone but also phenomena on a smaller scale like the Asian monsoon can be identified in higher order EOFs. With all available hardware resources, the computation took around 24 h in total. To assess the scalability, we repeated the procedure above on a reduced size dataset considering five years of the observations and measured computation times for different available computational resources. Fig. 5 shows the achieved speedup factors of using \( n \) processors over one processor in the execution of the two dominant operations (steps 2 and 4). Both operations scaled relatively well with the number of processors up to a certain limit, though the growth in speedup of the construction of the data matrix \( \mathbf{X} \) and singular value decomposition (SVD) became slower with increasing processors. The speedup limit close to 16 is in line with the experimental setup as 16 database instances have been used.

4.2. Land use change monitoring on Landsat image time series

Analyzing time series of EO imagery is a standard method to detect and monitor changes in land use and cover. The breaks for additive season and trend (BFAST) approach (Verbesselt et al., 2010; Verbesselt et al., 2012) for instance analyzes trend and seasonal components in individual pixel time series of satellite-derived vegetation index observations for structural changes. Applications so far have focused on relatively small study areas using Landsat imagery (see for instance Verbesselt et al., 2010; DeVries et al., 2015; Schmidt et al., 2015) or moderately large areas...
using MODIS imagery (see for instance Lu et al., 2016; Watts and Laffan, 2014). Due to the higher spatial resolution (30 m vs. 250 m), Landsat imagery is usually preferred. Therefore, there is a need to enable change monitoring with Landsat data on a larger (national or continental) scale. In this study case we represent approximately 11 years (21 July 2003 to December 31 2014) of Landsat 7 imagery covering an area of around 325,000 km$^2$ (approximately the size of Norway) located in the southwest of Ethiopia (Fig. 7) as a three dimensional SciDB array and apply BFAST to monitor changes in normalized difference vegetation index (NDVI) pixel time series after 1 January 2010. The dataset sums up to 1975 scenes in total and comes from 12 tiles in two different map projections (longitudinal UTM zones 36 and 37 respectively). Fig. 7 shows the study region and used Landsat tiles. Scenes from adjacent swaths clearly overlap. To prevent overwriting any values at these regions, we represent all scenes in a three-dimensional array with daily temporal resolution, although the repeat interval of the satellite is 16 days. Compared to the first study case, a few modifications to the ingestion were needed: before applying gdal_translate, we used gdal_merge to combine files of the same scene but different spectral bands and gdal_warp to warp scenes to the same coordinate reference system. Image dates have been again extracted from scene identifiers. The GDAL driver then automatically inserted scenes to the correct locations in the three-dimensional target array. As illustrated in Fig. 6, the resulting array is very sparse. It has $49548 \times 47713 \times 4177$ or approximately $9.8 \times 10^{12}$ cells in total but only around $54 \times 10^6$ (i.e., 0.5%) of the cells contain values, and are stored on disk. This illustrates one more the advantage of a sparse storage model for big array-based EO analytics.

BFAST, which is written in R, iteratively fits models to time series. This is not trivial to implement with provided SciDB operations, as BFAST reuses a lot of R components, including complex functions for probability distributions used for significance testing. To move these computations to the data, we used the SciDB extension r_exec (Paradigm4, 2016a), which allows running R scripts on array chunks in parallel. In order to reuse the existing R implementation of the BFAST approach (Verbesselt et al., 2010) within SciDB, each chunk of the Landsat array needs to contain complete time series and may cover only a relatively small spatial area. Thus, we rearranged the array such that individual chunks contain $(lat, lon, time) = (64 \times 64 \times 4177)$ cells. The R script to apply change monitoring on all chunks in parallel required around 20 lines of code. This includes additional operations to rearrange sparse input to R time series objects and to parallelize monitoring with BFAST on multiple time series per chunk within R. After the script has been executed over all chunks, the one-dimensional result array was again reshaped to a two-dimensional spatial array with number of observations, date of detected change, and magnitude of detected change as attributes. Including preprocessing, running BFAST, and postprocessing to download results from SciDB, around 60 lines of R code were needed.

Processing the whole area with all computational resources took around 9 days. Due to the high resolution of the output file, Fig. 7 shows upscaled results. The full resolution change map is provided at Zenodo (see Appendix B). Results indicate that large areas in the study region might have undergone abnormal changes in land use within the monitoring period, though some follow-up analysis on the significance and how change magnitude values refer to particular types of changes (e.g., deforestation) must be performed to draw valid conclusions.

For the evaluation of scalability, we selected a small subset covering 5% of the total area. We varied the numbers of available processors and measured speed of the computations by counting how many of the pixel time series could be processed on average within a minute. Fig. 8 presents the results. Up to the number of running SciDB instances, processing speed increased close to linearly with available computational resources. With 12 CPU cores, it was possible to process 2.8 times more time series in the same amount of time than using only 4 CPU cores. Due to the use of R level parallelization within array chunks, increasing the number of CPU cores beyond 16 still reduced computation times. The decrease in computation times was however fastest up to the number of running database instances.

### 4.3 Combining historic climate model projections with satellite-based observations

To evaluate the scalability of in-database spacetime joins, we ingested climate model output for historical precipitation from the NASA Earth Exchange Global Daily Downscaled Projections (NEX-GDDP) (NASA, 2014) to SciDB and joined the resulting array with the previously created TRMM array (Section 4.1). Both datasets have the same spatial ($0.25^\circ \times 0.25^\circ$) and temporal (daily) resolution but different extents. The NEX-GDDP dataset originally comes as a set of NetCDF files, where each file has daily projections for a single year, experiment, model, and variable. We adapted the

---

**Fig. 4.** First 10 EOFs from 17 years of daily accumulated precipitation observations from the TRMM mission. The dataset is available at http://doi.org/10.5281/zenodo.218466.

**Fig. 5.** Achieved computational speedup in the precipitation EOF analysis. Array reshaping refers to constructing the data matrix $X$ from the original three-dimensional array whereas SVD refers to the actual computation of the EOFs in a singular value decomposition. Operations with minor effects on computation times have been omitted here. 16 database instances have been used.
ingestion procedure by adding one argument to `gdal_translate` such that individual commands insert a single band (one day) of a file only. Since we used historical projections of a single model (MPI-ESM), iterating over all bands and all NetCDF files then created a single three-dimensional array. To compare model projections with the TRMM dataset we then joined the NEX-GDDP and TRMM arrays by space and time, which included the following steps:

1. Trim arrays NEX_GDDP and TRMM such that they have same spatial (−50° to 50° latitude and −180° to 180° longitude) and temporal (1 January 1998 to 31 December 2005) extents.
2. Derive indexes of array cells from NEX_GDDP, which correspond to cells in TRMM and add these indexes as new attributes to the original precipitation array TRMM.
3. Reshape the resulting array to the schema of TRMM such that computed index references become new dimensions, whereas old dimensions are dropped.
4. Combine attributes of this array with attributes of TRMM in a cell-wise join. The result contains attributes from both input arrays, i.e. model projected and satellite-derived precipitation values.

This procedure can be applied in a single nested SciDB query. Step 2 uses the `eo_over` operation from our extensions whereas other steps use standard SciDB functionality. To evaluate the scalability of such in-database joins, we concentrated on nontrivial steps 2 and 3 and measured computation times with a varying number of processors as in previous study cases.

Fig. 8 shows the achieved scalability in joining the climate model output with satellite-derived observations. Both considered operations turned out to be significantly faster with more processors. Speedups basically scaled linearly up to the number of running database instances. For the array reshaping, the speedup slightly increased even beyond, reaching its maximum at a 20 times higher speed with 32 CPU cores. Compared to the `eo_over` operation, reshaping arrays makes use of parallelization even within individual database instances. For both operations, we expect to achieve superior absolute speedup values for a larger number of running database instances. Compared to the scalability among database instances, the scalability within instances seems to be limited. Running many database instances thus seems to be advantageous even on single machines as in this experimental setup.
5. Discussion

With the case studies on EOF analysis, pixel time series classification, and image fusion we demonstrated how large Earth Observation datasets can be analyzed in a scalable way, by using open source tools. These case studies can be seen as particular instances of the larger analysis goal of information extraction or statistical learning from Earth Observation data (Cheng et al., 2017; Xia et al., 2017; Zhu et al., 2017). In this discussion we will focus on the scalability, transferability, reproducibility and cost-effectiveness of the proposed technical solution.

5.1. Scalability

Computations in the study cases scaled well with available processors up to the number of running database instances. We did not test the scalability in a cluster environment but results are promising and motivate to set up larger EO analytics platforms where, following Taft et al. (2014) and Mehta et al. (2017), we expect similar results. A comparative benchmark study on available systems including SciDB, xarray, Rasdaman, and GeoTrellis would be extremely useful to make conclusions on absolute performance differences, i.e. under which circumstances a system is advantageous over others. Furthermore, the overall performance and computation times of the presented approach still highly depend on parameters like the array partitioning schema, which must be chosen with regard to particular problems. As such, time series analyses clearly benefit from a temporal chunk organization, but array reshaping can be costly. In other words, the scalability suffers when there is no locality between the data and computations (Taft et al., 2014). For specific problems, we acknowledge that ad hoc file-based solutions may lead to superior results (Bakcsa et al., 2016; Liu, 2014). However, to achieve reasonable scalability, such solutions often require reimplementation of the algorithms with regard to massive parallelization, external memory optimizations, and data distribution.

5.2. Transferability

We are confident that the scalability in computations applies to other data-parallel algorithms. Problems that are hardly parallelizable or algorithms with rather global data access patterns, however, will not scale on their own. This for instance applies to many hydrological operations such as computing accumulation flow from digital elevation models. Compared to file-based EO storage solutions, the conversion of scenes to multidimensional arrays during ingestion implies additional computations and data transfers. The presented approach thus might be unrewarding for use as a data store only but may pay off in typical modeling or method development workflows where the data is ingested only once and used repeatedly in different models, methods, or with varying parameters. The built-in operations of SciDB, as in other data management systems, cover relatively little functionality in data analysis. Compared to similar systems, SciDB supports in-database linear algebra routines and running R or Python scripts as well as any other executable on a chunk-level parallelism. Both features turned out to be essential for the study cases but also facilitate other EO specific analytics and statistical modeling.

To flexibly interface with many EO datasets, we built a workflow around GDAL that transfers two-dimensional file-based scenes to multidimensional arrays. We demonstrated in a study case that this procedure can in principle be applied to datasets from climate model output too. Due to the GDAL model being only two-dimensional with additional bands, representing such higher-dimensional data (e.g. with latitude, longitude, elevation, experiment, representative concentration pathway, and experiment identifier as dimensions) can be cumbersome. As such, combining output from many models, experiments, and RCPs in a single array would require additional work. Solutions that directly work with respective higher-dimensional NetCDF of HDF files or extending the GDAL model to multiple dimensions could be potential future improvements.

5.3. Reproducible EO analytics

Although available EO datasets are such a valuable source for modeling and understanding processes on Earth, complex analytics with these data is often only hardly reproducible. To our knowledge, Google Earth engine (GEE) (Gorelick et al., 2017) currently is the only platform where researchers can easily share and collaborate in analyses. Full reproducibility, however, requires openness of all analysis procedures. In contrast to GEE, the presented approach in principle allows full reproduction since all used software components including our software extensions are available as open-source. To reproduce study cases of this paper on small datasets, we provide the sample code to automate setting up the required infrastructure using Docker (see Appendix B). For larger setups in cluster environments, this becomes more difficult and requires support by data centers. However, the very general and flexible representation of EO datasets as multidimensional arrays could help to abstract from particular infrastructures and software, such that analyses carried out at one data center could be shared and repeated at another data center. Standardized interfaces that disguise computing backends would strongly facilitate the reproducibility and thus increase the value of conclusions based on EO data. Developing such interfaces is under active research and development (Pebesma et al., 2017; Nativi et al., 2017).

5.4. Cost-effectiveness considerations

The gain of scalability and the ability to reproduce and share EO analytics come with the cost to set up the database, to load the data, and to familiarize users with the array data model. However, as soon as data volumes exceed local storage or processing capabilities, modelers need to revise their data management in any case. Compared to other scalable computing infrastructures like Hadoop, the array model in our opinion makes understanding the distribution of the data and processing much easier. Additionally, the suggested approach already comes with efficient linear algebra routines and interfaces to R and Python. Since this can highly reduce the amount and effort of reimplementation needed, this makes it a reasonable trade-off in many cases.
6. Conclusions

In this paper, we address challenges in analyzing large Earth observation (EO) datasets that are provided as collections of file-based scenes. We present a flexible approach and software tools using the geospatial data abstraction library to build spatiotemporal arrays from many scenes in combination with the data management and analytical system SciDB to scale computationally intensive analytics. The approach has been applied in scalable land-use change monitoring, global precipitation pattern analysis, and integration of climate model output with satellite-based observations. The considered datasets all could be converted to multidimensional arrays in a relatively simple procedure and the scalability of considered computations turned out to be promising. The speedup of computations mostly grew close to linearly with the number of available processors up to the number of running database instances. Required additional work to setup the database and to ingest the data still must be balanced with the gain of scalability and consider particular applications. Especially in statistical modeling, where methods are already available e.g. in R, the approach can facilitate scalability with relatively little amount of reimplementation needed. Since all used software is available as open-source, the presented approach also addresses difficulties in reproducing analytics based on EO data.

Appendix A. Computing infrastructure and experimental setup

All experiments have been performed on a Dell PowerEdge R815 Server with 4 AMD Opteron 6376 CPUs, i.e., 64 CPU cores in total and 256 GB main memory. The same SciDB installation as described in Table 2 has been used. To limit available computational resources, experiments have been performed within a Docker container which has been restricted to use a certain number of available processors only. All computation time measurements are median values over 5 repetitions.

Appendix B. Code and data availability

Source code of the developed tools is available at GitHub (see https://github.com/appelmar/scidb-geo and https://github.com/appelmar/scidb4gdal respectively). Supplementary material to this paper contains a demonstration how to run two of the study cases on small subsets of the original data. Therefore, we provide a Docker image that contains all needed software and data. Analyses take around an hour each on normally equipped desktop computers. It is available online at GitHub (https://github.com/appelmar/scidb-eo-ispresdemo). Full instructions how to build the Docker image and how to run the study cases can be seen in the provided README document. Results of the study cases on full-size datasets have been published at Zenodo and can be downloaded at http://doi.org/10.5281/zenodo.215975 and http://doi.org/10.5281/zenodo.218466 respectively.

Table 2

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of physical servers</td>
<td>1</td>
</tr>
<tr>
<td>Number of SciDB instances per server</td>
<td>16</td>
</tr>
<tr>
<td>Max. number of threads per query (per instance)</td>
<td>4</td>
</tr>
<tr>
<td>Cache size for array chunks (per instance)</td>
<td>1 GB</td>
</tr>
<tr>
<td>In-memory array threshold (per instance)</td>
<td>1 GB</td>
</tr>
<tr>
<td>Sort buffer (per instance)</td>
<td>256 MB</td>
</tr>
</tbody>
</table>

References


