GSio: A programmatic interface for delivering Big Earth data-as-a-service

Pablo R. Larraondo, Sean Pringle, Jian Guo, Joseph Antony and Ben Evans
National Computational Infrastructure, Australian National University, Canberra, Australia

ABSTRACT
In this paper, we present GSio, a software system for serving geospatial raster or gridded Big Earth Data at scale. GSio allows different scientific communities to consume geospatial analysis ready data. It provides a generic interface to the data, which removes the need to interact with individual files, and can interoperate with existing geospatial collections hosted on data centres and public clouds. A distributed compute model is used to read and transform the data in parallel using a cluster of compute nodes for delivering data as a service to users. Several use cases are presented demonstrating different scenarios where this service has been used.

1. Introduction
The term Big Earth Data (Guo, Wang, Chen, & Liang, 2014; Guo, Wang, & Liang, 2016) has been used to refer to large volumes of Earth observation data collected from satellites. This term has also been used to refer to any available geospatial data, including ground-based sensors and large computational model systems such as Numerical Weather Prediction (NWP) (Wagemann, Clements, Figuera, Rossi, & Mantovani, 2017). Platforms serving Big Earth Data face the added challenge of not only having to store, manage and process large collections of data, but they also need to provide interfaces for users to access the data.

Big Earth Data are used by different geospatial scientific communities, with different ways of manipulating, representing and presenting these data. For example, over the last decade, the climate community has invested significant effort in developing OPeNDAP data servers (Hyrax Data Server (West et al., 2011), Unidata Thematic Real-time Environmental Distributed Data Services (THREDDS) (Domenico, Caron, Davis, Kambic, & Nativi, 2006). Similarly, the environmental and geoscience communities have adopted Open Geospatial Consortium (OGC)-enabled web services, such as GeoServer (Deoliveira, 2008). There are a large number of international sites that have made data available through these services, allowing their users to visualise and download subsets, aggregations or whole data-sets. We discuss the use of different servers in Section 2.
There have also been activities that work across the discipline boundaries, and thus provide new potential to improve science and its outcomes. For example, the EarthServer project (Baumann et al., 2016) has successfully shown how climate and earth observation data can be delivered using a common OGC standards-compliant data service.

End-users who need to use Big Earth Data often require specific knowledge about each data collection. Different communities have developed their own tools, software and workflows around specific data collections which are often locked into specific configurations, file formats and map projections. Covering all the different possibilities is a challenging exercise, which requires cooperation from the different communities and consensus around standards.

The National Computational Infrastructure (NCI) developed the National Environmental Research Data Interoperability Platform (NERDIP) platform (Evans et al., 2015; Wyborn & Evans, 2015) which takes a broad approach to Big Earth Data. This platform enables interoperability of the data in a transdisciplinary way – using common data standards and conventions to harmonise the data. NERDIP enables programmatic in situ access to the data using domain-specific tools and ensures the data can be exposed using various community-driven data services.

There is critical need for data services to effectively access large volumes of data and satisfy users requirements. Different geospatial scientific domains have disparate ways to manipulate these data and there is a need for common Application Programing Interfaces (APIs).

Although the existing data services continue to provide a viable method for serving geospatial data to users, they have not been designed to scale to large data-sets. We note that we use the term geospatial data to refer to either geospatial gridded or raster data. Users now wish to perform exploratory data analysis by both visualising and interacting with on-demand generated data. They wish to combine different data sources, perform operations on the data, from (sub/up)sampling and data aggregation, to map re-projections, scaling or other operations. There is a further paradigm shift where users apply even complex algorithms, and it is no longer efficient to pre-compute and store derivative data products.

In data science, the term “data wrangling” has become popular for referring to the process of locating data and transforming it into a format that is useful for the user. This process is often more time-consuming than the final analysis of the data. In the case of geospatial data, users can spend a significant amount of time and effort creating appropriate representations of the data before the desired analysis can be performed. For example, the temporal and spatial extents contained by one file are normally different to the extents of the required analysis or computation.

These new activities place demands on data systems, due to the computational, I/O intensive and storage issues. The ready availability of compute in the current multi-core era (Michailidis & Margaritis, 2016) opens the possibility for generating on-demand products and performing interactive analysis on the data.

In this work, we propose GeoSpatial input/output (GSio) both as a common interface to interact with geospatial data, as well as a distributed data server that implements such interface. GSio presents a generic data model which is able to represent geospatial data, by abstracting away the concept of files to its users. To achieve this abstraction, GSio uses an indexing system which stores metadata from individual files such as temporal and spatial bounds.

We also use a scalable distributed compute model in which the process of reading and transforming data is offloaded onto a cluster of computational and I/O nodes to achieve scalable performance. GSio is then able to perform aggregations, map re-projections and
transformations to the data behind the scenes, so data can be presented to the user in a convenient form.

GSio delivers fast access to geospatial data using a common interface to represent geospatial data. Our objective is to engage with different communities, so existing workflows can be replicated or new ones built benefiting from both a performant and simplified access to the data.

This paper is structured as follows. Section 2 contains an introduction to the concept of geospatial data-as-a-service, diving into its challenges and reviewing alternative existing proposals. Section 3 describes the interface exposed by GSio, presenting a common data model as well as the method for requesting data. Section 4 presents the idea of applications making use of this service to target specific applications or communities. Section 5 presents real use cases where GSio has been demonstrated to work. The paper concludes with Section 6 providing a summary of the key contributions of this work and reflecting on future lines of research for this project.

2. Geospatial data-as-a-service

Applying the concept of Data-as-a-Service (DaaS) (Zheng, Zhu, & Lyu, 2013), we use the term Geospatial DaaS to describe the ability to deliver geospatial products to users on-demand.

Typical offline batch geospatial processing jobs distribute computation across individual data files generating new products or performing statistical analysis (Lewis et al., 2017). However, the content of an individual file rarely matches the requirements of users wanting to make use of the data. Often, users perform non-trivial processes on the data in order to be meaningful. There is a mismatch between the data as they are stored on filesystems and the representation needed by users in their workflows.

Geospatial DaaS provides a platform for geospatial analysis to be readily consumed by clients. With this approach, the complexities around accessing and transforming data are abstracted and managed as a server-side process. The server side focuses on delivering a high-performance service, while the clients-side performs analysis or visualisations relying on a harmonised data service.

2.1. The compute vs. storage trade-off

The notion of Geospatial DaaS raises the economic benefits of computing products on-demand. Derived products are commonly stored adjacent to the raw source data, often with significant storage overheads. Some of these derived products can be computed by performing an inexpensive operation on the original data. This leads to store more than one copy of the same collection based on different file formats or levels of aggregation being stored. The rationale behind this is often related to supporting different communities or use cases which require the data to be presented in a specific format. Due to the size of some geospatial collections, the economic cost of duplicating data-sets or storing derived products is significant.

Using published costs from one of the main public cloud providers, we can perform a comparison between the on-demand compute model vs. the pre-compute and store model. One Petabyte of data stored in a cloud-based object store costs around $300 K US dollars per year (in 2017). On the same cloud environment, an equivalent investment provides 1000
dedicated CPUs running for a whole year (*without taking into account I/O-related access and read/write costs*).

Figure 1 represents the economic costs of both models. In the case of computing derived products on-demand, a fixed number of CPUs will offer a certain quality of service relative to the required computations and the number of users that need to be served. By working out the total compute required for a specific use case, the intersection with the storage cost line will determine the minimum data volume where on-demand computation of derived products becomes a viable option. It is worth noting that the comparison presented here is an oversimplification we use to introduce the concept of Geospatial DaaS. Providing a comprehensive method for comparing these two methodologies is beyond the scope of this publication.

### 2.2. Abstracting the filesystem

In order to successfully provide an abstraction over the data contained on a filesystem, a Geospatial DaaS server needs to be aware of the contents of the individual files it exposes. Filesystems normally store data as files or objects using a tree structure which helps organise and locate the data. Often, files located under the same directory are related to each other.

In this paper, we use the concept of a Geospatial Collection as thematically related data with a defined geographical and temporal extent. Using a filesystem’s tree structure, we can relate a Geospatial Collection to the files contained within directories. A directory path could be used as the unique identifier for a particular element of a Geospatial Collection.

A Geospatial Collection can contain one or more variables which are related to each other, such as spectral bands, physical variables and products. Depending on the file format of the files and the structure of the collection, variables can be identified as data-sets in a file (in the case of netCDF4 and HDF5), using specific well-defined naming conventions for the files or ancillary metadata.

The Geospatial Collection is represented using a geospatial index, which maps the collections and variables into individual file locations. Each of these files normally contains a portion of the whole Collection, which is usually constrained between certain spatial and temporal bounds. By indexing the extents contained by each file, along different dimensions, provides a very powerful representation of the data. Such an index can be used to identify individual files of the Geospatial Collection, which contains specific variables, within certain spatial and temporal ranges.

![Figure 1. An idealised storage and compute cost model for storing derivative products vs. computing on the fly, using a DaaS model.](image)
Our proposal of Geospatial Data-as-a-Service makes internal use of a geospatial index to provide a queryable and structured view of the files in a filesystem. Users can request data by specifying general parameters such as a collection name, variable name and spatio-temporal extents and the server, will be able to locate, read and transform the data.

### 2.3. Geospatial data model and API proposal

Serving geospatial data to clients requires the definition of a well-defined interface that specifies how data are requested and how results are presented to the client.

Geospatial raster or gridded data are normally represented by two-dimensional numerical arrays in which each value is uniquely georeferenced to a specific point or region on the Earth. The earth observation and the climate communities have come up with two similar proposals for representing raster or gridded data. The Geospatial Data Abstraction Library (GDAL) (Warmerdam, 2008) defines the GDAL Data Model to represent raster data. Similarly, the climate community has come up with a general model, called the Common Data Model (CDM) (Nativi, Caron, Domenico, & Bigagli, 2008) which covers netCDF, OPeNDAP and HDF5, and is implemented by several libraries and protocols notably THREDDS.

CDM is a comprehensive model compared to the one GDAL defines, offering options to internally structure the data in groups or variables. Both models contain the concept of Data-set, which serves as the container to represent the numerical values for an extent. CDM defines the Data-set as a generic n-dimensional array, as opposed to GDAL where it is limited to a fixed two-dimensional array with bands acting as a third dimension.

The ability to represent multidimensional geospatial data is beneficial when performing certain analysis. Change detection or time series analysis requires data to be accessed along its temporal dimension. Similarly, RGB image composition or computer vision algorithms (Krizhevsky, Sutskever, & Hinton, 2012) used in machine learning often require an extra dimension representing the image channels.

Our proposed data model is inspired by both the GDAL and CDM models and we refer to it as the Geospatial Data Model (GDM). Using the simplicity of the GDAL model, we have extended it with a version that can contain multidimensional data. As the proposed model is used to interact dynamically with geospatial data, we have obviated a significant part of the functionality provided by the CDM, which is intended for archival and storage use cases.

The proposed GDM is built around the concept of a **Data-set**. Our GDM **Data-set** is a container for representing an n-dimensional numerical array, representing underlying geospatial data. A **Data-set** can then be decomposed into several components as shown in Figure 2.

- **Array** is the generic container for an n-dimensional numeric array. An **Array** is defined by a **DataType**, which determines the internal numerical type used to represent the data, and a **Shape**, which specifies the dimensions along the different dimensions in the array using a list of integers.

- **Dimension** is used to provide labels to the indexes along the different dimensions of a multidimensional array. **Dimension** is implemented as a dictionary that maps one-dimensional **Arrays** containing the values corresponding to the coordinates of an axis to the dimension name.

The **CRS**, which stands for Coordinate Reference System, is used to associate a coordinate system to the geospatial data contained in an **Array**. Following the same convention used in GDAL, **CRS** uses an OGC WKT text representation to describe a coordinate system.
The GDM Data-set uses these structures to represent an n-dimensional array containing geospatial data. A valid Data-set must contain a Dimension structure specifying as many dimensions as the contained n-dimensional data Array. The shape of the Array in a Data-set must match the ones contained in the Dimension structure for each axis. The mapping between axis of the Array and Dimensions is given as a list of names in the Data-set structure.

The GDM provides a description of how data are represented by the proposed Geospatial DaaS platform. To complete the description of the interface, we need to define a model for clients to submit requests. We note that at the beginning of this section, we introduced the concept of index as a representation of geospatial data collections. Using this idea, a client's request can be specified by specifying values for the different parameters such as Collection, Variable and ranges around the spatial and temporal extents.

Figure 3 represents the Request model for our proposed Geospatial DaaS, detailing the structure of a request and the returned response. We note that users requesting data have to specify a spatial extent, map projection and size of the returned Data-set but do not need to consider the original projection and resolution of the original data.

In Figure 3, BBox specifies a bounding box and Resolution indicates the size of a pixel in using the units of the corresponding CRS. The Geospatial DaaS server takes care of performing the corresponding transformations. Aggregation specifies a name for custom aggregation methods performed server side to present a uniform view of the data. Subsequently, these aggregation methods need to be properly documented and presented to the API users.

### 2.4. Review of existing data servers

In this section, we review existing geospatial data servers using two categories. The first category corresponds to those solutions which provide a low-level abstraction into geospatial data, for which we use the term "array databases". The second category is formed by other
solutions which offer high-level standards, or programming models, to interact with geospatial data.

In the first category, there are two well-known systems implementing the concept of array databases, Rasdaman (Baumann, Dehmel, Furtado, Ritsch, & Widmann, 1998) and SciDB (Stonebraker, Brown, Zhang, & Becla, 2013). While Rasdaman is focused on serving geospatial rasters, SciDB has a generic interface to different types of scientific data. Both provide an abstraction over geospatial data as generic numerical multidimensional arrays.

Rasdaman and SciDB provide well-defined interfaces to query data using domain-specific declarative languages, such as RASQL and AQL. While well-defined interfaces allow the data to be consumed in a new convenient way, different scientific communities have dependencies in the form of existing software tools and processing pipelines which need to be supported. Array databases require geospatial data to be ingested using specific formats which is different to the input files containing the original data. Due to the increasing volume of geospatial collections, maintaining two separate copies of the same data rapidly becomes unfeasible.

In the second category, we cover a few different systems and platforms commenting on the models they expose to interact with the data.

The Earth observation community uses GeoServer as the main method for delivering raster data. Similarly, the climate community uses THREDDS to deliver gridded data. GeoServer implements the OGC WCS protocol and THREDDS serves data using the OpenDAP protocol. WCS, as implemented in GeoServer, requires the server to pre-compute pyramids before the data can be served. These pyramids speed up access to the data but are expensive to compute and double the storage requirements to serve a data collection. OpenDAP, on the other hand, is a protocol designed to serve data from individual files. It provides methods for subsetting and serialising data inside a file but lacks the ability to provide a higher level abstraction on a data collection level. THREDDS offers the possibility of defining some virtual aggregations but it is limited to the dimensions and the number of files that can be exposed under a single virtual file. The main limitation of these two data servers, when serving Big Earth Data, comes from the fact that they are designed to operate as single servers. When large volumes of data need to be processed and served, they are currently unable to decompose large requests into smaller ones and handle the distribution in the background. Their capabilities are bounded by the capacity of the single machine hosting the service.

Google Earth Engine (GEE) (Gorelick et al., 2017) is a cloud-based system which offers an public API and an associated web-based development environment for visualising and
analysing geospatial data at scale and on real-time. This system offers access to a large repository of climate and earth observation data-sets. The data are ingested using a specific format to allow efficient access and scalable processing capabilities. Although the API is documented in great detail, there are very few details about the implementation of the distributed architecture of the backend. This system cannot be used or deployed outside the cloud environment set up by Google.

GeoTrellis (Kini & Emanuele, 2014) is an open source project which implements a processing engine for geospatial data. The system is built using the Scala programming language and currently uses the Akka actor model (Haller, 2012) for distributed processing. GeoTrellis relies on the data being exposed using an HDFS filesystem with the individual files written using the GeoTIFF format. Although this is an actively developed project, at present it does not offer the possibility of exposing data in other file formats than GeoTIFF and filesystems different from HDFS.

3. A geospatial DaaS implementation

This section provides details about the implementation of a Geospatial DaaS server as described in Section 2. Our implementation, called GSio, is based on previous work on a distributed geospatial server called GSKY (Larraondo, Pringle et al., 2017). GSKY is a distributed geospatial data server which exposes WMS and WPS services. GSio shares the same code base with GSKY but provides a low-level interface to the data. GSio provides a base service for both GSKY and other data services, which act as consumers of geospatial data. This model allows for building simplified services or applications that benefit from efficient access to geospatial data, and hiding complexities such as accessing disparate file formats and data projections.

3.1. Distributed compute model

MapReduce (Dean & Ghemawat, 2008) has been a successful high-level model for parallel processing and Big Data analytics for the last decade. Implementations of this model, such as Hadoop and Spark, are common in data centres today and its model has been proven to scale up to thousands of nodes.

MapReduce provides a distribution model for tasks where large computations can be decomposed into smaller fragments with no dependencies. These fragments are then distributed among a cluster of nodes to perform the required computations in parallel. After all the fragments have been computed, the results are then merged into a single result. Each compute node stores a small portion of the whole data-set on its local disk. The whole cluster can be seen as a distributed filesystem where data are spread among the different nodes. There are several implementations of MapReduce filesystems, such as HDFS (Borthakur, 2008), which allow MapReduce operations to use data locality and maximise performance.

While MapReduce is a very efficient model for parallel processing, its Map and Shuffle phases are quite complex and depend on specific filesystems to perform its computations. In other distributed filesystems, such as Lustre (Schwan, 2003) and the S3 object store (Palankar, Iamnitchi, Ripeanu, & Garfinkel, 2008), nodes storing the data and performing the computations are not the same and access to data is always done through the network.

The availability of fast interconnection links between nodes in data centres makes the concept of data locality less critical. Thus, being able to transfer data between nodes
efficiently simplifies significantly the design of distributed computing systems. Under this assumption every node of the cluster has complete visibility over the data and can perform computations on any of its parts.

We note there are many low-level programming models that allow coordinating activities among different processes that run concurrently in a distributed system. These models are commonly referred as Inter Process Communication (IPC). Remote Procedure Calls (RPC) is one of these models which is based on the concept of nodes executing functions on remote machines as if they were local. Most programming languages have libraries that implement this model allowing to design multi-process and multi-node systems.

gRPC (Google, 2017b) is an open source library for implementing RPC systems which is based on Protocol Buffers for interface definition and data serialisation, HTTP2 as the transport mechanism and allows cancellation, authentication and data-streaming.

In the case of having a cluster of nodes connected via a distributed filesystem, it is feasible to use gRPC to expose remote functions in each node. Using the Requests model API, as described in Section 2.2, gRPC clients can then be implemented. These clients can fully utilise a cluster by load balancing function calls asynchronously among the nodes of the cluster. These functions access data from either a classic distributed POSIX filesystem such as Lustre or distributed object stores such as AWS S3. Figure 4 represents the distributed compute model used to implement GSio.

### 3.2. Metadata attribute search (MAS)

In Section 2.2, we introduced the concept of providing an abstraction over a filesystem or object store. This abstraction is based on the idea of having a geospatial index containing a structured representation of the hierarchical structure of POSIX filesystems or object stores, as well as the metadata of its contents. The purpose of this index is to quickly identify individual files or objects by concepts such as their collection, variable name and spatial and temporal extents.

To implement this geospatial index, we have chosen PostgreSQL, which is a well-known relational database system, in combination with the PostGIS extension that adds support for geographic objects.

To populate the contents of the database, we created a stand-alone programme which recursively traverses parts of a filesystem opening and extracting the metadata of each file or object that it contains. We refer to this programme as a “crawler” since it crawls and extracts...
all the relevant metadata contained therein. The crawler uses GDAL to open and extract metadata, as it offers a generic platform for accessing many geospatial file formats.

Depending of the number of files under the selected directory or buckets, the process of crawling can be time-consuming. To accelerate this, crawlers are executed in parallel using concurrent processes on multiple nodes. The crawler outputs a series of JSON documents containing details about the file location and metadata such as its spatio-temporal extents and variable names. Once the crawler has completed, the resulting JSON documents are ingested to populate the database with a representation of the contents of the data collection. The crawler needs to be run periodically, depending on the updates of the filesystem, to keep a consistent representation.

A RESTful API is used to expose an interface to the data stored in the index. This RESTful interface specifies a list of parameters such as the collection path, variable name, temporal and spatial ranges, which are encoded as part of a URL. These parameters are then used to compose an SQL query sent to the Postgres database. The result of this query is a list of files, which is returned to the client as a JSON-encoded document.

The indexing system, called Metadata Attribute Search (MAS), has been designed to process and serve high volumes of metadata in near real-time. A production instance is able to process queries in milliseconds, even for the ones comprising large spatial areas or temporal ranges, which often results in thousands of files or objects being identified. The database has been tuned to use indexes and materialised views to achieve this level of performance. To handle scalability and data growth, the contents of the database can be sharded by the different geospatial collections or by splitting collections into non-overlapping geographical extents.

### 3.3. GSio

GSio, which stands for GeoSpatial input/output, is the core module that builds on the aforementioned indexing service and distributed compute model. GSio implements the Geospatial DaaS server using the Geospatial Data Model and interface described in Section 2.3. GSio provides an abstraction layer over the underlying files that contain the data.

The design of GSio is inspired by the concept of flow-based programming (Morrison, 2010), in which data get transformed by processes that are connected forming a Directed Acyclic Graph (DAG). This programming model, allows having predefined processes, with well-defined inputs and outputs, which can be treated as black boxes, connected and reused at different parts of the graph. The processing model used in GSio is also related to stream processing (Abadi et al., 2003), which introduces parallelism by defining compute units that are connected in a graph and can run concurrently. Data in this model are “streamed” through the network and computations at each stage happen concurrently improving performance and latency.

GSio is implemented using the Go programming language (Pike, 2012). Go provides a programming model that simplifies writing concurrent programmes. There are several proposals for building concurrent processing pipelines using this language (Google, 2017a; GopherAcademy, 2017). In the case of a geospatial DaaS, the concept of composable processing pipelines allows for the definition of efficient data processing models. Before a homogeneous array containing geospatial data can be presented to the user, the server needs to perform a series of processes such as locating files, reading subsets, re-projecting, resampling and merging which can expressed as a processing pipeline.

Figure 5 presents an example on how a request for data, geospatial data, for the southeast of Australia is handled by GSio. Different processes in the pipeline perform different actions to
generate the result. An incoming request to the pipeline defines some spatial and temporal boundaries for a certain collection and variable as well as the resolution and map projection for its output. To produce the result, different files containing data for this request have to be identified, read, transformed and merged into a numeric array. Each of these steps can work concurrently and their results are passed to the next stage using Go channels to improve parallelism.

In this model, the minimum unit of computation that circulates through the pipeline is a file. The process of reading data from disk is distributed using a cluster of nodes directly connected to the data to maximise performance.

The model that GSio presents can be scaled to adapt to serve different levels of demand or volumes of data. The size of the cluster used to read the data can be tuned on demand using a load balancer to distribute the load created by the pipeline. Also, as pipelines are dynamically created to serve each request, different nodes can be used in parallel to serve the requests.

GSio is exposed as a gRPC service by implementing the GDM model, which is directly translated into the Protocol Buffers language (version 3) specifying the input and output structures as well as the RPC call to request a geospatial Data-set. gRPC also provides efficient access to the data, which is serialised using protocol buffers. Ten of the main programming languages are officially supported by the gRPC project; hence, implementing GSio clients to consume the data for different environments is straightforward.

Figure 6 represents the tiers in which different communities can interact with geospatial data. GSio provides a common interface for the data enabling the definition of new services that comply with specific standards to deliver the data to specific users or platforms.

4. Use cases

The Geospatial DaaS introduced in Section 3 has been used to serve geospatial data in multiple ways at the NCI. This section provides some examples of different applications that make use of the proposed system using different interfaces and data collections.

Figure 5. Representation of the geospatial processing pipeline implemented in GSio.
4.1. GEOGLAM RAPP

The GEOGLAM Rangeland and Pasture Productivity (RAPP) initiative from the Group on Earth Observations (GEO) provides practical tools to monitor the condition of the world’s rangelands and pasture lands and to assess their capacity to sustainably produce animal protein.

Using MODIS satellite imagery, a worldwide fractional cover product, containing the three vegetation fractions (photosynthetic, non-photosynthetic and bare-soil), is available from the year 2000 up to the present (Guerschman et al., 2015). The spatial resolution of this collection is approximately 500 metres and eight days in time. This collection is stored as netCDF4 files containing the original sinusoidal projection and extents from the upstream NASA MODIS data-distribution, but stacking one year’s worth of data along the temporal dimension.

This data are exposed as OGC WMS and WPS services to the user community. Different WMS layers, such as an RGB composite of the three fractions, total cover and NDVI, can be computed and served on the fly from the original data using GSio. Similarly, a WPS service is provided to analyse temporal evolution of the vegetation for user-defined polygons (see Figure 7).

4.2. Australian intertidal extents model

Geoscience Australia has published the InterTidal Extents Model (ITEM) data-set (Sagar, Roberts, Bala, & Lymburner, 2017), which outlines the spatial extents of the exposed intertidal zone for all coastal regions in the Australian continent. Over 30 years’ worth of Landsat images were processed using an automated pipeline relating each image to its tidal height using a median-based compositing technique. The data-set is being used by Australian state government to assess coastal land cover risks and to study the habitats of migratory shorebirds.
Figure 7. Screen capture of the GEOGLAM RAPP map instance.

Notes: The map corresponds to the total vegetation cover index and the lower part represents a time series plot of precipitation and vegetation fractions (Photosynthetic, Non-Photosynthetic and Bare-soil) for a user-defined region in the Namibian rangelands.
Figure 8. Intertidal Extents Model (ITEM) WMS layer, showing the intertidal extents at the coast near Derby, Western Australia.
The data are presented as a collection of netCDF4 files containing tiles of the Australian coast using the Australian Albers projection and 25 metres resolution. These data are served as WMS layer creating an aggregated mosaic that can be presented to the user. GSio presents the raw data corresponding to each tile of the WMS layer, at the demanded resolution and projection. These arrays can then be scaled using a colour ramp and encoded as a PNG image which is returned to the WMS client. Clients can move around the data-set zooming in and out and the service is able to provide the required data on-demand (see Figure 8).

### 4.3. Machine learning on the ERA-Interim climate reanalysis

The machine learning community has recently demonstrated impressive results in the area of computer vision using methods based on convolutional and recurrent neural networks. The same techniques can be applied to raster or gridded geospatial for tasks such as feature detection (Iglovikov, Mushinskiy, & Osin, 2017), image classification (Li, Tao, Tan, Shang, & Tian, 2016) and even interpreting the contents of images into natural language (Lienou, Maître, & Datcu, 2010).

Several generic libraries have been published recently, facilitating the introduction of these techniques to new fields. TensorFlow (Abadi et al., 2016) is an example of these libraries which presents a comprehensive Python environment to work with neural networks.

In this use case, we use GSio as an interface to present climate data from the ERA-Interim (Dee et al., 2011) climate model as Python Tensor objects, which are used to train neural networks (Larraondo, Inza, & Lozano, 2017). A simple Python GSio client was implemented to provide an interface to the data and convert the Array objects into Python Numpy arrays.

In this work, the geopotential height parameter at three different levels of the atmosphere was used to derive observed precipitation at several locations in Europe. Users of this module can directly request climate modelled data by specifying the desired variable name and the temporal and spatial extents (see Figure 9).

### 5. Conclusions and future work

In this paper, we present GSio, which provides an interface to expose raster or gridded geospatial data as well as an implementation of a server that can be distributed and scaled adapting to demand. The main benefit of this model is to provide users with the ability to
By providing a well-defined low-level interface to the data, different applications and services can build on top to serve specific communities or use cases. In this sense, we have demonstrated how data can be exposed under different standards using this model, such as WMS, WPS and Python Tensors. Our goal, when designing this system, was to offer a model to the different scientific communities, such that specific geospatial applications can be built by facilitating the access to the underlying data.

Although the initial implementation of this service was carried out in the context of a High-Performance Computing (HPC) environment, with a fast-interconnected distributed filesystem, we intend to evolve the same model into a generic framework. Commercial cloud providers currently offer a data access model, based on object store, with interesting scaling capabilities in terms of the input/output operations. Further work needs to be done to adapt the model to this environment and produce comparisons between the differences in performance.

An active field of research around this project is also finding new ways of indexing geospatial data (GitHub.com, 2017). Techniques, such as space filling curves and bitmap database indexes, have the potential of improving the efficiency for locating data in a filesystem. New trends in cloud storage, such as HDF Cloud (HDF-Group, 2017), are based on reducing the size of the storage units, which requires scalable systems used to index the data.

Overall, we aim to work with various end-users from diverse scientific communities to help review and validate the usefulness of the proposed interface and data model in order to develop an interoperable, generic model that flexibly interacts with Big Earth Data.

Data availability statement

The data that support the findings of this study (https://doi.org/10.4225/25/5760F5452D371 and https://doi.org/10.5067/MODIS/MCD43A4.006 and https://doi.org/10.1002/qj.828) are openly available at the National Computational Infrastructure.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

The authors wish to acknowledge funding from the Australian Government Department of Education, through the National Collaboration Research Infrastructure Strategy (NCRIS) and the Education Investment Fund (EIF) Super Science Initiatives through the National Computational Infrastructure (NCI), Research Data Storage Infrastructure (RDSI) and Research Data Services (RDS) Projects.

References

Abadi, D. J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., … Zdonik, S. (2003). Aurora: A new model and architecture for data stream management. The VLDB Journal – The International Journal on Very Large Data Bases, 12(2), 120–139.

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., … Devin, M. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.
Baumann, P., Dehmel, A., Furtado, P., Ritsch, R., & Widmann, N. (1998). The multidimensional database system RasDaMan. *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, 575–7. Seattle, Washington: ACM.

Baumann, P., Mazzetti, P., Ungar, J., Barbera, R., Barboni, D., Beccati, A., … Calanducci, A. (2016). Big data analytics for earth sciences: The earthserver approach. *International Journal of Digital Earth, 9*(1), 3–29.

Borthakur, D. (2008). HDFS architecture guide. *Hadoop Apache Project S3*.

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM, 51*(1), 107–113.

Dee, D. P., Uppala, S. M., Simmons, A. J., Berrisford, P., Poli, P., Kobayashi, S., … Bauer, P. (2011). The ERA-interim reanalysis: Configuration and performance of the data assimilation system. *Quarterly Journal of the Royal Meteorological Society, 137*(656), 553–597.

Deoliveira, J. (2008). GeoServer: Uniting the GeoWeb and spatial data infrastructures. Paper presented at the Proceedings of the 10th International Conference for Spatial Data Infrastructure, St. Augustine, Trinidad.

Domenico, B., Caron, J., Davis, E., Kambic, R., & Nativi, S. (2006). Thematic real-time environmental distributed data services (thredds): Incorporating interactive analysis tools into nsdl. *Journal of Digital Information, 2*(4).

Evans, B., Wyborn, L., Pugh, T., Allen, C., Antony, J., Gohar, K., … Wang, J. (2015). The NCI high performance computing and high performance data platform to support the analysis of petascale environmental data collections. Paper presented at the International Symposium on Environmental Software Systems.

GitHub.com. (2017). S2 geometry library in go. Retrieved October 18, 2017 from https://github.com/golang/geo

Google. (2017a). Go concurrency patterns: Pipelines and cancellation. Retrieved October 18, 2017 from https://blog.golang.org/pipelines

Google. (2017b). grpc.io. Retrieved October 18, 2017 from https://grpc.io/

GopherAcademy. (2017). Composable pipelines improved. Retrieved October 18, 2017, from https://blog.gopheracademy.com/advent-2015/composable-pipelines-improvements/

Gorelick, N., Haner, M., Dixon, M., Ilyushchenko, S., Thau, D., & Moore, R. (2017). Google earth engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*.

Guerschman, J. P., Held, A. A., Donohue, R. J., Renzullo, L. J., Sims, N., Kerblat, F., & Grundy, M. (2015). The GEOGLAM rangelands and pasture productivity activity: Recent progress and future directions. Paper presented at the AGU Fall Meeting Abstracts.

Guo, H., Wang, L., Chen, F., & Liang, D. (2014). Scientific big data and digital earth. *Chinese Science Bulletin, 59*(35), 5066–5073.

Guo, H., Wang, L., & Liang, D. (2016). Big earth data from space: A new engine for Earth science. *Science Bulletin, 61*(7), 505–513.

Haller, P. (2012). On the integration of the actor model in mainstream technologies: The scala perspective. Paper presented at the Proceedings of the 2nd edition on Programming systems, languages and applications based on actors, agents, and decentralized control abstractions.

HDF-Group. (2017). HDF-cloud. Retrieved from https://www.hdfgroup.org/hdf-cloud/

Iglovikov, V., Mushinskii, S., & Osin, V. (2017). Satellite imagery feature detection using deep convolutional neural network: A Kaggle competition. *arXiv preprint arXiv:1706.06169*.

Kini, A., & Emanuele, R. (2014). Geotrellis: Adding geospatial capabilities to spark. Spark Summit.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Paper presented at the Advances in neural information processing systems.

Larraondo, P. R., Inza, I., & Lozano, J. A. (2017). Automating weather forecasts based on convolutional networks. In *Proceedings of ICML Workshop on Deep Structured Predictions*.

Larraondo, P. R., Pringle, S., Antony, J., & Evans, B. (2017). GSKY: A scalable, distributed geospatial data-server. In Proceedings of the Academic Research Stream at the Annual Conference Locate, Research@Locate 2017, co-located with Digital Earth & Locate 2017 1913 (pp. 7–12).

Lewis, A., Oliver, S., Lymburner, L., Evans, B., Wyborn, L., Mueller, N., … Sixsmith, J. (2017). The Australian geoscience data cube – Foundations and lessons learned. *Remote Sensing of Environment*.
Li, Y., Tao, C., Tan, Y., Shang, K., & Tian, J. (2016). Unsupervised multilayer feature learning for satellite image scene classification. *IEEE Geoscience and Remote Sensing Letters, 13*(2), 157–161.

Lienou, M., Maitre, H., & Datcu, M. (2010). Semantic annotation of satellite images using latent Dirichlet allocation. *IEEE Geoscience and Remote Sensing Letters, 7*(1), 28–32.

Michailidis, P. D., & Margaritis, K. G. (2016). Scientific computations on multi-core systems using different programming frameworks. *Applied Numerical Mathematics, 104*, 62–80.

Morrison, J. P. (2010). *Flow-based programming: A new approach to application development: CreateSpace*. Retrieved from https://dl.acm.org/citation.cfm?id=1859470

Nativi, S., Caron, J., Domenico, B., & Bigagli, L. (2008). Unidata’s common data model mapping to the ISO 19123 data model. *Earth Science Informatics, 1*(2), 59–78.

Palankar, M. R., Iamnitchi, A., Ripeanu, M., & Garfinkel, S. (2008). Amazon S3 for science grids: A viable solution? Paper presented at the Proceedings of the 2008 international workshop on Data-aware distributed computing.

Pike, R. (2012). Go at Google. *Proceedings of the 3rd annual conference on Systems, programming, and applications: Software for humanity, 5–6*. Tucson, Arizona: ACM.

Sagar, S., Roberts, D., Bala, B., & Lymburner, L. (2017). Extracting the intertidal extent and topography of the Australian coastline from a 28 year time series of Landsat observations. *Remote Sensing of Environment, 195*, 153–169.

Schwan, P. (2003). Lustre: Building a file system for 1000-node clusters. Paper presented at the Proceedings of the 2003 Linux symposium.

Stonebraker, M., Brown, P., Zhang, D., & Becla, J. (2013). SciDB: A database management system for applications with complex analytics. *Computing in Science & Engineering, 15*(3), 54–62.

Wagemann, J., Clements, O., Figuera, R. M., Rossi, A. P., & Mantovani, S. (2017). Geospatial web services pave new ways for server-based on-demand access and processing of Big Earth Data. *International Journal of Digital Earth, 1–19*.

Warmerdam, F. (2008). The geospatial data abstraction library. *Open Source Approaches in Spatial Data Handling, 87–104*.

West, P., Fox, P. A., Gallagher, J., Potter, N., Holloway, D., & Zednik, S. (2011). OPeNDAP Hyrax: An extensible data access framework within the Earth System Grid Federation. Paper presented at the AGU Fall Meeting Abstracts.

Wyborn, L., & Evans, B. J. K. (2015). Integrating ‘Big’ geoscience data into the petascale national environmental research interoperability platform (NERDIP): Successes and unforeseen challenges. Paper presented at the Big Data (Big Data), 2015 IEEE International Conference.

Zheng, Z., Zhu, J., & Lyu, M. R. (2013). Service-generated big data and big data-as-a-service: An overview. Paper presented at the Big Data (BigData Congress), 2013 IEEE International Conference.