RDPro : Distributed Processing of Big Raster Data

[Scalable Data Science]

Zhuocheng Shang
Samriddhi Singla*
Ahmed Eldawy
Computer Science and Engineering
University of California, Riverside
{zshan011,ssing068,eldawy}@ucr.edu

Elia Scudiero
USDA-ARS, U.S. Salinity Laboratory
Department of Environmental Sciences
University of California, Riverside
Riverside, CA, USA
elia.scudiero@ucr.edu

ABSTRACT

Advancements in remote sensing technology allowed for collecting vast amounts of satellite and aerial imagery with up to 1 cm pixel resolutions, stored in raster format crucial for various research fields. However, processing this data poses challenges, including resolving data dependencies when location, resolution, and coordinate systems do not align and managing large datasets within memory constraints. This paper introduces RDPro, a novel Spark-based system that efficiently processes and analyzes large raster datasets. RDPro features a new data model tailored for data dependencies in a distributed, shared-nothing environment, complete with tools for loading and writing raster data. It also optimizes core raster operations within Spark, allowing users to integrate complex data science workflows. Comparative analysis shows RDPro outperforms existing systems by up to two orders of magnitude.

PVLDB Reference Format:
Zhuocheng Shang, Samriddhi Singla, Ahmed Eldawy, and Elia Scudiero. RDPro : Distributed Processing of Big Raster Data. PVLDB, 18(3): 613 - 622, 2024.
doi:10.14778/3712221.3712229

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ZhuochengShang/RDPro-Big-Raster-Data-Processing.

1 INTRODUCTION

Advancements in remote sensing technology have led to an ever-increasing amount of geospatial data. In the past few years, over 500 satellites have been launched [51], collecting Earth observational data at resolutions from 50 cm to 1 km per pixel. Today we have petabytes of earth observational data which became an important component of research in fields such as disaster response and monitoring [5, 21], wild fire detection [41], management of energy and natural resources [12, 32, 33], agricultural monitoring [11, 20, 24, 37, 39, 55], and marine biology [19, 27]. Machine learning paves the way for more application including object detection, classification and prediction [1, 8, 20, 24].

*Work done while at UCR, currently works at Meta

Proceedings of the VLDB Endowment, Vol. 18, No. 3 ISSN 2150-8097.
doi:10.14778/3712221.3712229

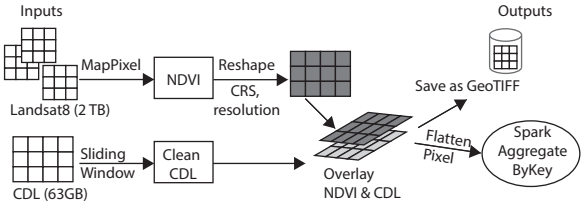


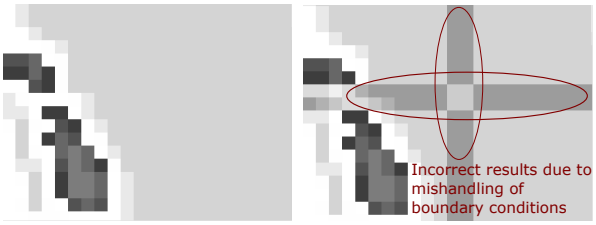
Figure 1: Example of RDPro Query Pipeline

Figure 1 illustrates an example of a query pipeline that an agronomist can use to track crop health using satellite imagery. First, it loads the Cropland dataset (CDL) [38] and applies a *sliding window* operation that cleans the data. Concurrently, it loads the Landsat8 dataset [46] and computes the normalized difference vegetation index (NDVI) for each pixel. To combine both datasets, we *reshape* Landsat8 to match the coordinate reference system (CRS) and resolution of CDL. Then, we *overlay* both datasets. This output is *saved as GeoTIFF* files that the agronomist can further explore in a GIS software. Additionally, the individual pixel values can be *flattened* to aggregate using a standard Spark *aggregateByKey* operation.

The above example highlights three main challenges that data scientists face when processing raster data. First, the system should handle a single large file as in CDL and tens of thousands of small files as in Landsat8. Second, it should process and optimize complex query pipelines that mix raster operations, e.g., sliding window and reshape, and regular operations, e.g., aggregateByKey. Third, raster data processing often requires geographical alignment of thousands of files to ensure a correct answer.

There are many big spatial data systems that can handle vector data [2, 15, 52, 54, 57] but only limited work on raster data [16, 29, 40, 59]. Some systems tried to systematically process big raster data, e.g., GeoTrellis [18], Rasdaman [4], Google Earth Engine [23], Apache Sedona [56], and others [17, 34, 44, 59]. However, these systems suffer from one or more limitations that hinder their use in real data science applications as further detailed below:

(1) **Single Machine:** Some traditional database systems and programming packages [17, 34] for raster data are limited to a single machine and either fail or take too long to process large-scale data.
(2) **Heavy Ingestion:** Some raster-based systems [4, 23, 44] require an expensive data ingestion to load data into an internal model for partitioning and indexing, optimizing for repetitive queries, while modern applications often focus on one-time ad-hoc queries.



(a) RDPro (Correct) (b) GeoTrellis (Wrong)

Figure 2: Boundary conditions of convolution operation

(3) **Limited Functionality:** Some systems [7, 9, 30, 56] support limited pixel-level operations but lack functionality for processing the dataset as a coherent unit. For example, some systems ignore tile boundaries in some operations. Figure 2 illustrates an example of the convolution operation where GeoTrellis produces wrong results at tile boundaries while RDPro handles the data correctly. This illustrates the risk of prioritizing performance over accuracy.
(4) **Constrained Query Runtime:** GDAL and ChronosDB [17, 59] are designed to run one primitive operation at a time, and users have to run each operation separately.
(5) **Expensive Memory Usage:** GeoTrellis and Apache Sedona [18, 56] load large data chunks into memory, causing failures when the dataset exceeds memory capacity.

This paper proposes *RDPro*, a novel distributed system implemented on Spark that helps data scientists to efficiently perform complex analysis on big raster data. It overcomes the limitations of existing systems as follows: (1) RDPro is a distributed system built on Spark. It introduces a new concept of *Maplets* by extending Spark RDD as data processing model. (2) RDPro avoids data ingestion step and it directly process raster files in common formats including GeoTIFF, NetCDF, and HDF. (3) RDPro implements an exhaustive list of operations that users may need to analyze raster data. (4) RDPro extends the Spark RDD model to support raster data which gives the users the advantage to run a complex spatial query pipeline on their datasets. (5) RDPro has built-in components to partition data into small units that can fit in memory.It also uses memory optimization strategies to reduce memory cost along operations. We run an extensive experimental evaluation that compares RDPro to the state-of-the-art Spark-based systems, GeoTrellis [18] and Sedona [56]. RDPro has up-to two orders of magnitude performance gain while being perfectly able to scale to big raster data. Through our collaboration with environmental science, we provide case studies of RDPro applied in real applications.

The rest of this paper is organized as follows: Section 2 covers the related work. Section 3 describes the proposed data model and the design of raster operations in RDPro. Section 4 runs experimental evaluations of the proposed system. Section 5 describes three real-world use cases and how RDpro can provide benefits. Finally, Section 6 concludes the paper.

2 RELATED WORK

This section covers the relevant work in the area of raster data processing including single-machine and parallel systems.

2.1 Single Machine

The GIS community makes use of tools such as QGIS [35], ArcGIS [36], PostGIS [34], GDAL [17] and raster analysis packages in R [25] and Python [22] for their application needs. However, all of these systems are limited to a single machine making them inefficient when working with large raster datasets. When compared to distributed systems, these single-machine systems either fail to process large datasets or take significantly more time to perform queries on large raster datasets. The proposed system *RDPro* is implemented in Spark, which makes it more efficient and scalable for processing large raster datasets than single-machine systems.

2.2 Parallel Raster Systems

Parallel raster processing systems can be categorized into CPU-based distributed systems and GPU-based parallel systems. **CPU-based** distributed systems that can process raster data include SciDB [6, 44, 45], Rasdaman [3, 4], GeoTrellis [18], Apache Sedona [56, 58], Google Earth Engine [23], and ChronosDB [59]. Some systems use MapReduce such as SciSpark [30], SciHadoop [7], MrGeo [29], and ClimateSpark [9, 26]. The next part gives an overview of these distributed systems and framework, especially the fundamental operations, such as reprojection and file loading process.

Some distributed work [7, 9, 30, 31, 44, 49] use the array data model to support big scientific data and ignore the geographical component, i.e., coordinate reference system (CRS), of the raster data. They lack the reprojection operation which is particularly critical when dealing with datasets with mixed projections. It is common to store a big raster dataset as a set of files, called scenes, with mixed projections, e.g., Landsat8. Thus, ignoring the geographical component renders these system unusable for big raster data.

Some systems [4, 23] support geographical data, but are limited by an expensive data ingestion step. These systems implement their own data model and require an ingestion phase that reads the data and re-structures it according to their data model. Other systems [59] require the user to manually place the files on each machine in a specific way for the operations to work. This process is suitable for repetitive queries on the same dataset where the cost of the ingestion step is amortized over the number of queries. However, modern data science applications often require one-time ad-hoc queries which makes the data ingestion step a bottleneck.

Some systems [56, 59] support limited raster operations and are not suitable for most data science applications. Furthermore, they suffer from memory issues [18, 56] as they require loading large data blocks into memory before processing. This limits their scalability as shown in this paper.

GPU-based approaches are natural for processing raster data and there is some preliminary work to utilize GPUs in raster analysis. Some work [13, 14] focuses on vector data processing on GPU which is typically done through rasterization. Other raster-based work addresses specific raster operations such as indexing[28, 61, 62], compression[28, 61, 62] and polygon rasterization[48, 50, 60]. Some work [43, 53] provides raster processing frameworks but are limited in reprojecting multiple files with different CRS. Work[53] focuses on improving primitive operations using GPU while writing intermediate data to disk. Map reprojection work [42] also faces similar limitations in handling multiple file dependencies with their

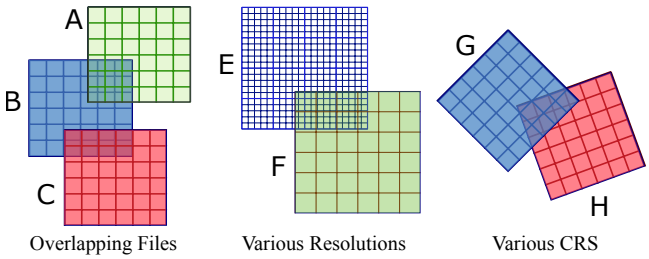


Figure 3: Challenges with Real-world Raster Data

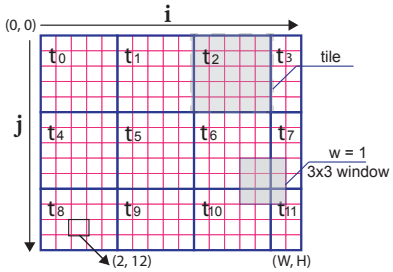


Figure 4: A raster dataset with width $W = 17$ and height $H = 14$ partitioned into a grid of 5×5 pixels.

GDAL-based design and offers less functionality compared to this paper. In general, this paper focuses on disk-based distributed processing so GPU processing can be integrated to improve in-node performance.

The proposed system, *RDPro*, is a distributed raster system that fully supports geographical operations. It works directly on raster files as they are obtained from satellite data repositories with no need for any data conversion or ingestion. RDPro provides a rich processing runtime that includes the major raster operations including reprojection which is necessary to combine multiple datasets together. RDPro takes complex query pipelines from users, breaks it down into small units, and process it in parallel. This keeps memory usage under control while scaling to terabytes of data.

3 PROBLEM FORMULATION AND DESIGN

First, we describe the challenges of handling real-world raster data. Second, we define the raster data. Next, we define the logical data model utilized in this system design and operations for raster analysis. Finally, we present memory and network optimization solutions.

3.1 Problem Setting: Real Raster Data

Real world raster generally comes from various sources including raw satellite data, rectified data, aerial imagery, and machine-generated raster products such as land cover maps. Data scientists explore these datasets to monitor land cover, crop health, or climate change. One research demand requires the combination of multiple data sources, e.g., Sentinel-2 and Landsat-8, for better performance and coverage [7, 29]. However, the harmonization methodology becomes increasingly complex as more raster data sources are used. We summarize these challenges in the following few points which could all occur at the same time.

(1) **Multiple overlapping files:**. In general, a single raster dataset can be made available as a large set of files, sometimes thousands of files. For example, Figure 3 shows files A, B, and C which could belong to one dataset. Sometimes, these files slightly overlap which results in conflicting data that need to be resolved. Thus, one cannot simply treat the dataset as a single large array. Yet, these small arrays need to be aligned correctly to process overlapping or boundary regions.
(2) **Various resolutions:** Raster products can have a wide range of resolutions depending on the sensor used to collect the data, e.g., E & F in Figure 3. As a real example, MODIS data has 1 km resolution while some Sentinel products have 10 m resolution. This means that a single pixel of MODIS can overlap 10,000 Sentinel pixels. Users will want to harmonize these datasets.
(3) **Various CRS:** A CRS defines a geographical projection that maps a region of the earth surface to a two-dimensional space that corresponds to a raster dataset. Various datasets could have different CRSs such as G&H in Figure 3. Even one raster product could have multiple CRS depending on the geographical location, e.g., UTM zones in Landsat data. To combine these datasets, users need to easily reproject datasets to a common reference space before processing them.
(4) **Large scale size:** Satellite images can be collected daily, with up-to terabytes of data with high-resolution. Additionally, satellite images are dense, resulting in larger file sizes.

3.2 Raster data definitions

The first step in building a new query processing engine for raster data processing is to define a logical data model to represent the data and operations on it.

Definition 1 (Grid Space, G). *A grid space is defined as a two-dimensional grid that consists of W columns and H rows with the origin $(0, 0)$ at the top-left corner as shown in Figure 4. The location of a raster in the grid space bears no resemblance to what geographical area it represents.*

Definition 2 (Pixel, p). *A pixel $p = (i, j)$ represents the cell in the grid at column $0 \leq i < W$ and row $0 \leq j < H$.*

Definition 3 (Raster Tile). *Given a grid G, and tile dimensions $tw \times th$, the grid is broken down into tiles where each tile consists of tw columns and th rows, with the exception of the tiles at the last column and/or the last row. Additionally, Figure 4 illustrates an example of a raster tile with associated tile ID.*

Definition 4 (Measurement, M). *The measurement is a function that defines a value for each pixel. We use $M(i, j)$ to indicate the value of the pixel at location (i, j).*

Definition 5 (Non-geographical Raster Dataset). *A non-geographical raster dataset is defined by a grid G and a measurement function M.*

Definition 6 (World Space). *The world space represents a rectangular space on the Earth's surface defined by four geographical coordinates (x_1, y_1) and (x_2, y_2) that define the space $[x_1, x_2[\times[y_1, y_2[$.*

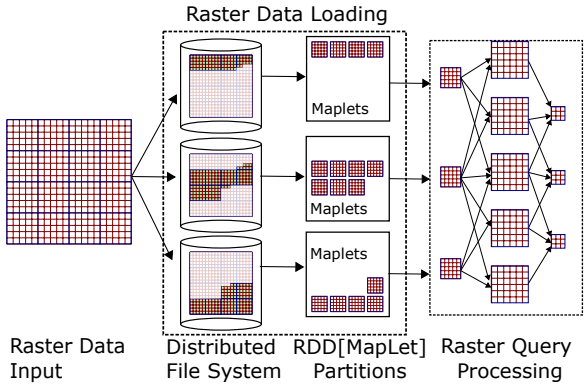


Figure 5: RDPro Architecture of RDD[Maplet]

The world space is associated with a coordinate reference system (CRS) that maps world coordinates map to earth's surface.

Definition 7 (Grid-to-World, $\mathcal{G}2\mathcal{W}$). *$\mathcal{G}2\mathcal{W}$ is a 2D affine transformation that transforms a point from the grid space to the world space. The inverse of this matrix is called world-to-grid, $\mathcal{W}2\mathcal{G} = \mathcal{G}2\mathcal{W}^{-1}$ and can be used to map locations from world space back to grid space.*

Definition 8 (Geographical Raster Dataset, R). *A geographical raster dataset is defined by a grid space, $G = (W, H)$, a measurement function M, a grid-to-world $\mathcal{G}2\mathcal{W}$ transformation, and a CRS defined by a unique spatial reference identifier (SRID).*

In a geographical raster dataset, termed *raster dataset* from this point on, each pixel occupies a rectangular space in the world defined by transforming its occupied grid space using the associated $\mathcal{G}2\mathcal{W}$. The measure value $M(i, j)$ of that pixel indicates a physical value measured for that area, e.g., temperature or vegetation. Although, the pixel width and height in grid space is one unit of measurement, in world space the pixel width may not be equal to pixel height. For example, the pixel width in world space may be 80 *cm* and height may be 30 *cm*.

3.3 RDPro Design with Raster Operations

This section outlines RDPro design and defines how the data model is integrated into Spark. We then detail how it addresses real-world raster data challenges through a pipeline of various operations.

Figure 5 illustrates the RDPro **system design**. To efficiently process raster data in a distributed environment, RDPro needs to split large raster datasets into smaller pieces that can be processed independently. At the same time, it should be able to treat arbitrarily large raster datasets as one coherent dataset. To accomplish this goal, RDPro introduces the notion of ***Maplets***. A Maplet is a subset of the raster data with additional location information, termed *MapLocator*, enabling RDPro to process it independently. While many file formats and systems split raster files into *tiles*, they remain tied to the parent raster file. Maplets can freely move between machines to allow greater scalability in a shared-nothing system. Maplets also benefit from reducing memory usage because they are a small subset of the data, which is different from loading the data as an array in other works [18, 57].

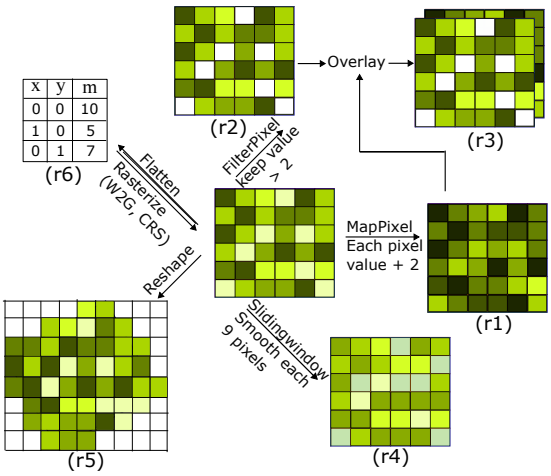


Figure 6: Raster Operations

Definition 9 (Raster MapLocator (RM)). *Each raster is associated with auxiliary information called MapLocator, which is a lightweight record that consists of the following information: (1) Raster grid size, W and H. (8 bytes) (2) The raster grid-to-world, $\mathcal{G}2\mathcal{W}$, transformation. (48 bytes) (3) The SRID of the raster CRS. (4 bytes) (4) Tile width and height, tw and th. (8 bytes)*

Definition 10 (Raster Maplet). *A Maplet consists of a MapLocator, a tile ID (t_{id}), and measure values for this tile.*

The innovation of RDPro is to redefine all raster operations to process a set of Maplets which allows them to scale on large clusters and process terabytes of data.

Parallel data loading is essential in a distributed system. RDPro reads both large files and a directory with numerous of files. It splits each file into 128 MB partitions, aligning Spark's lazy execution paradigm. Once the entire query pipeline is defined, see Figure 1, each partition is assigned to a processing core. Each core first reads the file header to extract MapLocator information. To ensure correct reading of the tiles across partition boundaries, each processing core reads the tiles that start at the corresponding partition and attaches the MapLocator to each of them, as illustrated in Figure 5.

Raster operations in RDPro define the basic set of operations that data scientists use to build a query pipeline. All these operations are defined as Spark transformations that can be integrated with existing Spark operations into one coherent application. Due to limited space, this part focuses on three operations, *Reshape*, *SlidingWindow*, and *Overlay*. Other operations, e.g., MapPixels or FilterPixels, Flatten, and Rasterize, are simpler and are excluded from this discussion.

(1) *Reshape(R_1, RM_2)*: The reshape operation, as in Figure 6(r5), converts the input raster dataset to a target MapLocator RM_2 with possibly different CRS, raster size, and tile size. This operation is crucial in aligning one or more raster datasets.

One challenge with the Reshape function is the complex conversion between coordinate reference systems involving non-linear functions like trigonometric functions. To solve this, the Reshape

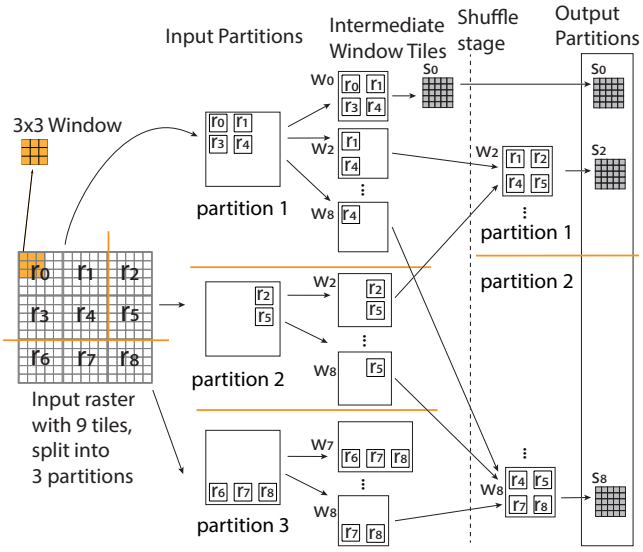


Figure 7: Overview processing of SlidingWindow function

operation computes a grid-to-grid $\mathcal{G}2\mathcal{G}$ transformation to map locations from the input Maplet to the output grid space.

To calculate $\mathcal{G}2\mathcal{G}$ for a specific source Maplet r_{id}, RDPro begins by retrieving a transformation function $\mathcal{W}2\mathcal{W}$ that transforms between the world coordinates of r_{id} and the target CRS of RM_2. Some existing libraries, e.g., GeoTools and Proj4J, provide this function for all standard CRS definitions. After that, the $\mathcal{G}2\mathcal{W}$ from the two MapLocators, $\mathcal{G}2\mathcal{W}_1$ and $\mathcal{G}2\mathcal{W}_2$, are used to compute $\mathcal{G}2\mathcal{G}$ as:

$$\mathcal{G}2\mathcal{G} = \mathcal{G}2\mathcal{W}_2^{-1} \circ \mathcal{W}2\mathcal{W} \circ \mathcal{G}2\mathcal{W}_1$$

, where $\circ$ denotes function composition. RDPro will optimize the composite function, e.g., if $\mathcal{W}2\mathcal{W}$ is an affine transformation, then the final function $\mathcal{G}2\mathcal{G}$ is optimized by multiplying the three affine matrices into one matrix. Subsequently, the inverse function $\mathcal{G}2\mathcal{G}^{-1}$ is calculated similarly. With $\mathcal{G}2\mathcal{G}$, the reshape operation continues by processing the Maplet r_{id} to map all its pixels to the target raster. This creates a set of partial Maplets which are then grouped by their IDs and merged into a set of final Maplets.

(2) *SlidingWindow*(R, w, f): As shown in Figure 6(r4), this operation applies a user-defined window calculation function to generate an output raster dataset. For $w \geq 1$, this focal operation needs all pixels within a $2w+1 \times 2w+1$ window around each pixel. Figure 7 shows an example when $w = 1$, there are $3 \times 3 = 9$ pixels in the window. A key challenge is efficiently handling pixels at Maplet boundaries, especially, when spread across different machines.

A naive implementation of SlidingWindow would work at the *pixel level*. For $w = 1$, each input pixel is replicated to nine output windows which incurs a huge computational and memory overhead that makes it unscalable.

To solve the above challenge, we develop an optimized algorithm that works at the *Maplet level*. It replicates each input Maplet to the affected output Maplets, using shallow copies to minimize overhead. We also introduce intermediate *WindowTiles*, a logical wrapper that groups input Maplets contributing to one output Maplet. For example, in Figure 7, w_0 contains the four input tiles that contribute

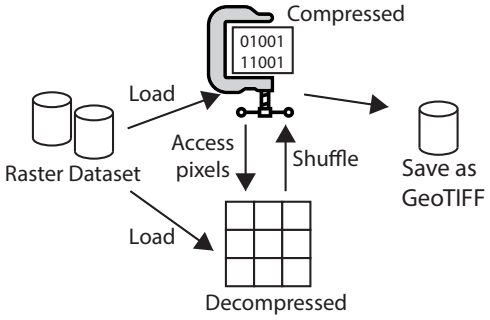


Figure 8: Maplet compression life cycle along network

to the output Maplet s_0. Once the WindowTile is complete, e.g., w_0 with four tiles, the output tile is eagerly computed. Otherwise, we group the window tiles by their ID to compute the final set of tiles as shown in Figure 7. The eager computation of output Maplets greatly reduces the network overhead since we only shuffle one final output Maplet instead of up-to nine input Maplets.

(3) *Overlay*(R_1, R_2): The overlay operation stacks R_1 and R_2 and returns a single RDD[Maplet] as shown in Figure 6(r3). The Overlay operation ensures that each Maplet in the datasets shares the same MapLocator. With the help of *Reshape*, the implementation of Overlay becomes simpler and more efficient. Once the two datasets are aligned, we co-partition and group them by t_{id} and create a wrapper that stacks the two tiles. This operation is used to combine multiple bands per pixel including time series data.

Data writing efficiently produces raster files on disk. While parallel data writing into multiple files is straightforward, RDPro merges them into a single file for GIS applications. GeoTrellis and Sedona collect all pieces into a single machine, causing crashes with large GeoTIFF files. RDPro introduces **compatibility mode** overcomes this. This mode operates in two phases: the first involves parallel data writing. Each worker node independently writes its Maplets to a file. It keeps track of the offset and length of each tile on disk. The second phase concatenates all the files into a single file and updates the tile offsets in the concatenated file. Accordingly, it compiles the final header of the output file that allows a standard GIS software to read the file correctly.

3.4 Memory and Network Optimization

RDPro employs two main memory optimization strategies. First, to reduce the memory and network overhead RDPro employs the lifecycle shown in Figure 8 to automatically compress and decompress Maplets as needed. When a file is initially loaded, Maplets are represented in the same state of the input data which is usually compressed. A Maplet is lazily decompressed when pixels are accessed. It is compressed again when it is shuffled over network or when the final output is written to disk.

Second, objects in Spark are immutable so we must create a new Maplet to modify pixel values, e.g., multiply each value by a constant. RDPro addresses this by creating a lightweight *tile-wrapper* class that references the original Maplet and adds the user-defined function, e.g., multiply by a constant. Values are then converted

Table 1: Raster Datasets

Dataset	# pixels	Resolution	Size
CDL_Riverside	52.71 M	30 m	201.09 MB
CDL_California	1.57 B	30 m	5.33 GB
CDL_US	16.88 B	30 m	62.88 GB
Landsat8_Riverside	237.97 M	30 m	689.86 MB
Landsat8_SoCal	421 M	30 m	1205 MB
Landsat8_SoCal+Central	542 M	30 m	1551 MB
Landsat8_California	1.57 B	30 m	4.40 GB
Landsat8_US	52.53 B	30 m	146.78 GB
Landsat8_World	798.47 B	30 m	2.18 TB
Planet_US	2.46 T	3 m	6.7 TB

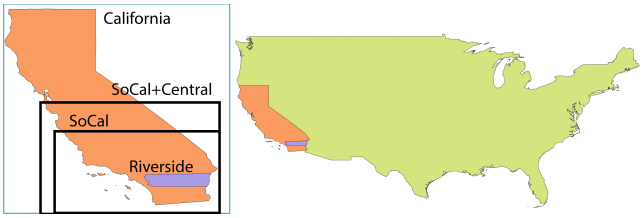


Figure 9: California and US Landsat8 Dataset Coverage

on-the-fly as needed. This approach benefits many operations, such as convolution calculations.

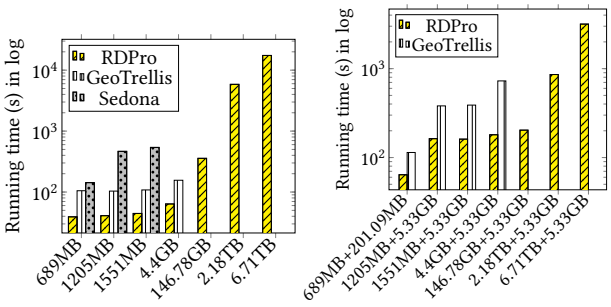
4 EXPERIMENTS

This section provides an experimental evaluation that compares RDPro to the distributed systems, Apache Sedona [56] and GeoTrellis [18]. We did not compare RDPro with Google Earth Engine due to its unsuitability for systematic and reproducible evaluations, as it lacks hardware control and has unstable run times influenced by user demand. Additionally, Rasdaman was excluded from the comparison because its public version, running on a single machine, does not offer a fair comparison with distributed systems.

4.1 Setup

We run *RDPro*, GeoTrellis, and Sedona on a Spark 3.1.2 cluster with one head node and 12 worker nodes. The head node has 128 GB RAM, 2×8 core processors (Intel(R) Xeon(R) E5-2609 v4 @ 1.70GHz), and each worker node has 64 GB RAM with 2×6 core Xeon processors on CentOS Linux. We compare GeoTrellis-Spark 3.6.3 and Apache Sedona 1.5.0, running each experiment three times and averaging the results. Experiments with GeoTrellis and Sedona increase executor memory to 36GB, up from the default 16GB, to help with their memory issues, while RDPro uses default settings.

Table 1 lists the datasets used in the experiments and their attributes. All raster datasets except Planet Data are publicly available. The CDL [38] and Landsat8 [46] dataset are made available by the US Department of Agriculture (USDA) and US Geological Survey (USGS), respectively. Planet_US is sourced from Planet Labs [47]. The size in bytes of each dataset reflects the number of bands and data type of measurement values. In the Figure 9, we visualize Landsat8 datasets to provide a clearer overview of the relationships between the datasets.



(a): Load+Write **(b): Overlay**

Figure 10: Load+write, Overlay on RDPro and baselines

4.2 Data Loading and Writing

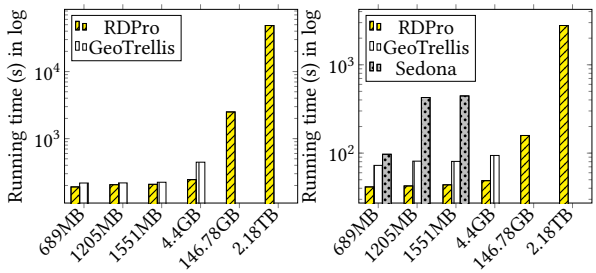
This experiment loads each dataset and writes it back as a new GeoTIFF file to test loading and writing performance. Figure 10(a) shows the running time of both the loading and writing steps combined for datasets that range from 700MB to 6.7TB. Comparing baselines, first, RDPro is consistently faster than baselines with up-to 3x speedup than GeoTrellis. Second, existing systems fail to load and/or write large datasets as they tend to throw an out-of-memory exception. In particular, Sedona is designed to load each file as a single array and cannot partition large files. This fits its proposed design to load small files that represent individual features rather than processing large raster products [8]. On the other hand, GeoTrellis raster writer needs to stitch all the data into a single machine which causes failure even for moderately sized datsets, e.g., 150 GB. Medium-scale data does not experience a dramatic speedup because distributed systems perform better with large-scale data. In medium-scale cases, not all resources may be fully utilized. For instance, if the dataset is small, it may not be evenly distributed across all machines, leading to parallelization overhead without significant performance gains.

4.3 Overlay

This experiment evaluates the performance of the Overlay operation when combining two datasets: CDL with Landsat8 and CDL with Planet. These results include the time of the reshape operation that modified Planet or Landsat8 to match the MapLocator of CDL. Effectively, this adjusts the resolution, the CRS, and the tile size to perfectly align the two datasets. Sedona is excluded from this experiment due to the lack of a reproject function. To run this operation in GeoTrellis, it needs to collect all metadata centrally on a single machine to clip and align the two datasets which results in its poor performance shown in Figure 10(b). RDPro is about three times faster for small and medium datasets. As the data size grows, RDPRo continues to scale while GeoTrellis starts to throw several errors which shows its poor performance and lack of robustness.

4.4 Reshape Operation

This experiment evaluates the performance of the Reshape operation on RDPro and GeoTrellis in two scenarios, Reproject and



(a): Reproject **(b): Rescale**

Figure 11: Reproject and Rescale performance on Landsat8

Table 2: SlidingWindow running time in seconds on CDL

Data Size	201.09MB	5.33GB	62.88GB
RDPro	25.45	34.09	94.93
GeoTrellis	35.45	121.05	591.22

Rescale. In **Reproject**, we convert the CRS of Landsat8 datasets to the commonly used EPSG:4326 CRS. Sedona does not support this operation so it was excluded. We write the output to GeoTIFF to ensure that the computation of the final raster is complete. Figure 11(a) shows the performance of the Reproject operation. RDPro is almost twice as fast as GeoTrellis and can scale to the biggest dataset while GeoTrellis encounters out-of-memory errors and fails because it tries to load and decompress the raster tiles as an array which is limited to 2GB in Java. We notice that RDPro still takes a significant amount of time when processing large-scale data, and this can be optimized by utilizing a tile wrapper approach similar to the one employed in SlidingWindow. In **Rescale**, we reduce the resolution of the Landsat8 dataset to produce a lower-resolution version by reducing both width and height by a factor of 10, which will return a 100 times smaller raster. We also write the output to GeoTIFF to ensure all computations are complete. Similarly, for the Rescale operation in Figure 11(b), RDPro scales seamlessly and provides a better performance than GeoTrellis. The better performance of RDPro is despite the fact that GeoTrellis ignores boundary conditions and produces incorrect results as shown in Figure 2.

4.5 SlidingWindow (Convolution)

This experiment evaluates the performance of the SlidingWindow operation in RDPro and GeoTrellis. Sedona does not support this operation. We use a window of size one ($w = 1$) and calculate the average of all the nine pixels in the window. This experiment requires all tiles in the input dataset to have the same MapLocator. To focus on the performance of the SlidingWindow function, we only apply it on the CDL dataset because it ships as a single file. As shown in Table 2, both RDPro and GeoTrellis can process all the datasets but RDPro is up-to 6 times faster. This is an impressive result, especially that GeoTrellis has an unfair advantage of ignoring the difficult boundary cases as shown in Figure 2.

Table 3: Impact of SlidingWindow tile-wrapper on shuffle and execution memory on CDL

RDPro SlidingWindow	with tile-wrapper		without tile-wrapper	
Data Size	Shuffle Size (GB)	Peak Execution Memory (GB)	Shuffle Size (GB)	Peak Execution Memory (GB)
201.09MB	0.028	0.33	0.039	11.54
5.33GB	0.65	12.37	1.11	258.5
62.88GB	9.41	77.51	17.28	777.29

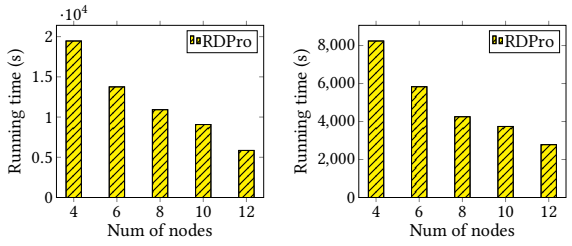
Table 4: Impact of tile compression on shuffle and execution memory on large scale dataset

FilterPixel + Reshape	Compressed		Decompressed	
Data Size	Shuffle Size (GB)	Peak Execution Memory (GB)	Shuffle Size (GB)	Peak Execution Memory (GB)
146.78GB	0.41	9.11	1.17	9.26
2.18TB	4.56	150.41	13.22	153.94
6.7TB	40.12	706.48	98.25	725.31

4.6 RDPro Memory and Network Usage Study

This experiment evaluates the impact of the WindowTile approach on the SlidingWindow operation. It demonstrates the effect of this work in optimizing memory usage and shuffle size during complex operations. We compare the operation using the WindowTile design with a pixel level implementation. As shown in the Table 3, the eager computation of the output inside the SlidingWindow tile-wrapper significantly improves performance. It cuts the shuffle size in half and reduces the peak memory usage by an order of magnitude. In Table 4, we show the effect of automatic tile compression during the shuffle operation. We first run FilterPixel and then call the Rescale operation as described in subsection 4.4. The results show that processing with compressed tiles along the network can reduce shuffle size by more than 60% and peak execution memory by around 2%. These optimizations significantly enhance performance and enable RDPro to handle large scale datasets.

4.7 RDPro Scalability Study



(a): Load+Write **(b): Rescale**

Figure 12: RDPro scalability study on Landsat8_USGS

To evaluate the scalability of RDPro, we vary the numbers of worker nodes from 4 to 12 with 12 cores per node in Figure 12. Our focus is on the Landsat8_USGS dataset, where we conduct

load, write, and rescale operations. From the figure we make two observations. First, RDPro scales linearly with the number of nodes which indicates good load balancing. Second, even with four nodes, RDPro is able to complete all experiments which is the result of the Maplet design and the memory optimizations.

5 CASE STUDIES

This section presents three case studies, covering applications in agriculture, visualization, and data preprocessing.

5.1 ET Model Calculation in Agriculture

This case study focuses on implementing the BAITSSS model [10, 11] to estimate the evapotranspiration (ET) from satellite and weather data. This application has real-world applications to address water shortage and promote water sustainability in agriculture.

The initial step harmonizes all datasets to match Landsat8 resolution, requiring the upsampling lower resolution rasters from 1000 m to 30 m. The model also performs hundreds of calculations, straining Python scripts due to limited memory.Consequently, intermediate results are written to and read back from disk, which complicates and slows down computations. Previous work [11] was limited to small areas of interest, such as individual crop fields.

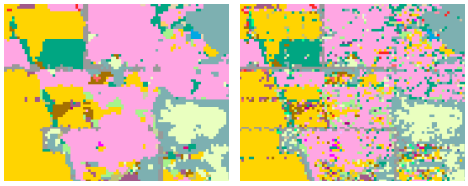
RDPro implements the ET-model and addresses memory limitations using key operations: **Reshape**, **Overlay**, and **MapPixel**. The process begins by reshaping all datasets for consistent resolution, CRS and geographical extent. Then, all rasters are stacked into a single RDD[Maplet] using the Overlay function, allowing MapPixel to perform calculations directly. Each cycle builds on previous results for time-series analysis, and the output is written as a single GeoTIFF. RDPro can process extensive areas, such as the entire California Central Valley farmlands for 24-hour analysis, in 4 hours. The current bottleneck is in the Reshape process. We plan to optimize it using a tile-wrapper similar to WindowTile. In contrast, the Python script takes 3.5 hours to upsample one raster, with the full pipeline requiring at least 82 hours for one day of data.

5.2 Raster Dataset Visualization

We tackle the real-world scenario of providing data scientists with a visualization or overview of a large dataset before downloading. We introduce a *thumbnail operation* using the **Reshape** function to create lower-resolution images. This approach combines all images in the dataset into a single coherent GeoTIFF with the same CRS, making it easily portable and quick to open in GIS software, e.g., QGIS. We import Landsat8_World data and compile it into a unified GeoTIFF file with a 1000 m pixel resolution. RDpro completes in 11 minutes, much faster than parallelized GDAL, which takes 8 hours. GeoTrellis encounters errors when reshaping to a customized pixel resolution. RDPro also complete Planet_US data in 58 minutes.

5.3 Data Preprocessing for ML/DL Pipeline

This case study explores the application of machine learning and deep learning models to satellite images, such as using SAM segmentation on agricultural imagery [24]. The model require cleaned-up satellite images, and as noted in the work [63], enhancing image quality is necessary due to differences between traditional visual and remote sensing imagery.



(a) Cleaned-up CDL (b) Original CDL

Figure 13: Cleaned-up and original CDL

We use CDL_California in this case study, applying a **SlidingWindow** function to smooth the data by assigning the majority value from a 9 element array to the center value. The comparison between cleaned-up CDL and the original one is shown in Figure 13. Subsequently, we write the output as a GeoTIFF file for further analysis. RDPro completes this process in 270 seconds. GeoTrellis lacks support for custom window functions, and Kernel or convolution calculations are unsuitable for categorical crop data.

This case study also extends to the query pipeline described in Figure 1. RDPro completes the query pipeline for the California region in 498 seconds and finishes the Riverside area in 22 seconds. In contrast, a Python script using *rasterio* package takes significantly longer and must write intermediate results to disk after each operation. The Python script completes the Riverside area in 1,458 seconds. Additionally, the Python script requires merging all files into a single image before feeding it into the pipeline, which further slows down the process. Finally, we use the overlay operation to combine the smoothed CDL with NDVI time series obtained from Sentinel-2 satellites [24]. The final data that we feed into the ML pipeline contains the crop at each pixel along with the NDVI time series for an entire growing season, e.g., one year.

6 CONCLUSION

This paper described *RDPro*, a distributed system for efficiently processing large raster data in data science applications. RDPro features the RDD[Maplet] data model, improving data loading, spatial queries, and file writing. Experiments confirm its scalability over GeoTrellis and Apache Sedona, with case studies demonstrating its effectiveness and pratical value. Future work will focus on implementing new features. Currently, RDPro only supports nearest neighbor and average interpolations, but it aims to include bilinear and bicubic methods in the future which are challenging as they require processing more input pixels for each output pixel. Additionally, RDPro is looking to improve upsampling, rescaling, and reprojection operations using a similar tile-wrapper approach used in sliding window and convolution.

ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation (NSF) under grant IIS-2046236 and by Agriculture and Food Research Initiative Competitive Grants no. 2020-69012-31914 from the USDA National Institute of Food and Agriculture.

REFERENCES

[1] Shantanu Aggarwal. 2022. *SATLAB-an End to End Framework for Labelling Satellite Images.* Ph.D. Dissertation. Arizona State University.

[2] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel Saltz. 2013. Hadoop-GIS: A high performance spatial data warehousing system over MapReduce. In *PVLDB*, Vol. 6.

[3] Peter Baumann. 2022. The Rasdaman Array DBMS: Concepts, Architecture, and What People Do With It. In *SSDBM*. Association for Computing Machinery, Article 27.

[4] Peter Baumann, Andreas Dehmel, Paula Furtado, Roland Ritsch, and Norbert Widmann. 1998. The Multidimensional Database System RasDaMan. In *SIGMOD*. Seattle, WA, 575–577.

[5] Daniel Brown et al. 2015. Monitoring and evaluating post-disaster recovery using high-resolution satellite imagery–towards standardised indicators for post-disaster recovery. *Martin Centre: Cambridge, UK* (2015).

[6] Paul G Brown. 2010. Overview of SciDB: large scale array storage, processing and analysis. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 963–968.

[7] Joe B Buck, Noah Watkins, Jeff LeFevre, Kleoni Ioannidou, Carlos Maltzahn, Neoklis Polyzotis, and Scott Brandt. 2011. Scihadoop: Array-based query processing in hadoop. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–11.

[8] Kanchan Chowdhury and Mohamed Sarwat. 2023. A Demonstration of GeoTorchAI: A Spatiotemporal Deep Learning Framework. In *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023*. ACM, 195–198.

[9] ClimateSpark 2018. ClimateSpark. https://github.com/feihugis/ClimateSpark

[10] Ramesh Dhungel, Robert Aiken, Paul D Colaizzi, Xiaomao Lin, Dan O'Brien, R Louis Baumhardt, David K Brauer, and Gary W Marek. 2019. Evaluation of uncalibrated energy balance model (BAITSSS) for estimating evapotranspiration in a semiarid, advective climate. *Hydrological Processes* 33, 15 (2019), 2110–2130.

[11] Ramesh Dhungel, Ray G Anderson, Andrew N French, Todd H Skaggs, Mazin Saber, Charles A Sanchez, and Elia Scudiero. 2024. Early season irrigation detection and evapotranspiration modeling of winter vegetables based on Planet satellite using water and energy balance algorithm in lower Colorado basin. *Irrigation Science* 42, 1 (2024), 15–27.

[12] Chris Dickens et al. 2019. Defining and quantifying national-level targets, indicators and benchmarks for management of natural resources to achieve the sustainable development goals. *Sustainability* (2019).

[13] Harish Doraiswamy and Juliana Freire. 2020. A gpu-friendly geometric data model and algebra for spatial queries. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 1875–1885.

[14] Harish Doraiswamy and Juliana Freire. 2022. Spade: Gpu-powered spatial database engine for commodity hardware. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2669–2681.

[15] Ahmed Eldawy and Mohamed F Mokbel. 2015. Spatialhadoop: A MapReduce Framework for Spatial Data. In *2015 IEEE 31st international conference on Data Engineering*. IEEE, 1352–1363.

[16] Ahmed Eldawy, Mohamed F Mokbel, Saif Alharthi, Abdulhadi Alzaidy, Kareem Tarek, and Sohaib Ghani. 2015. Shahed: A MapReduce-based System for Querying and Visualizing Spatio-temporal Satellite Data. In *2015 IEEE 31st international conference on data engineering*. IEEE, 1585–1596.

[17] GDAL/OGR contributors. 2022. *GDAL/OGR Geospatial Data Abstraction software Library.* Open Source Geospatial Foundation. https://doi.org/10.5281/zenodo.5884351

[18] GeoTrellis on Spark 2019. GeoTrellis on Spark. https://github.com/wri/geotrellis-zonal-stats/blob/master/src/main/scala/tutorial/ZonalStats.scala.

[19] Pierre Gernez, Stephanie CJ Palmer, Yoann Thomas, and Rodney Forster. 2021. remote sensing for aquaculture. *Frontiers in Marine Science* 7 (2021), 1258.

[20] Rahul Ghosh, Praveen Ravirathinam, Xiaowei Jia, Ankush Khandelwal, David J. Mulla, and Vipin Kumar. 2021. CalCROP21: A Georeferenced multi-spectral dataset of Satellite Imagery and Crop Labels. In *IEEE BigData*. IEEE, 1625–1632. https://doi.org/10.1109/BIGDATA52589.2021.9671569

[21] Thomas W Gillespie et al. 2007. Assessment and prediction of natural hazards from satellite imagery. *Progress in Physical Geography* (2007).

[22] Sean Gillies et al. 2013–. *Rasterio: geospatial raster I/O for Python programmers.* Mapbox. https://github.com/rasterio/rasterio

[23] Noel Gorelick et al. 2017. Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote sensing of Environment* 202 (2017), 18–27.

[24] Rutuja Gurav, Het Patel, Zhuocheng Shang, Ahmed Eldawy, Jia Chen, Elia Scudiero, and Evangelos Papalexakis. 2023. Can SAM recognize crops? Quantifying the zero-shot performance of a semantic segmentation foundation model on generating crop-type maps using satellite imagery for precision agriculture. arXiv:2311.15138 [cs.CV] https://arxiv.org/abs/2311.15138

[25] Robert J. Hijmans and Jacob van Etten. 2012. *raster: Geographic analysis and modeling with raster data.* http://CRAN.R-project.org/package=raster R package version 2.0-12.

[26] Fei Hu, Chaowei Yang, John L Schnase, Daniel Q Duffy, Mengchao Xu, Michael K Bowen, Tsengdar Lee, and Weiwei Song. 2018. ClimateSpark: An in-memory Distributed Computing Framework for Big Climate Data Analytics. *Computers & geosciences* 115 (2018), 154–166.

[27] Daniel Kachelriess, Martin Wegmann, Matthew Gollock, and Nathalie Pettorelli. 2014. The application of remote sensing for marine protected area management. *Ecological Indicators* 36 (2014), 169–177.

[28] Nathalie Kaligirwa, Eleazar Leal, Le Gruenwald, Jianting Zhang, and Simin You. 2014. Parallel QuadTree encoding of large-scale raster geospatial data on multi-core CPUs and GPGPUs. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data* (Dallas, Texas) *(BigSpatial '14)*. Association for Computing Machinery, New York, NY, USA, 30–39.

[29] Yi Liu, Luo Chen, Wei Xiong, Lu Liu, and Dianhua Yang. 2012. A MapReduce Approach for Processing Large-scale Remote Sensing Images. In *The 20th International Conference on Geoinformatics, Geoinformatics 2012, Hong Kong, China, June 15-17, 2012*. IEEE, 1–7. https://doi.org/10.1109/GEOINFORMATICS.2012.6270312

[30] Rahul Palamuttam, Renato Marroquín Mogrovejo, Chris Mattmann, Brian Wilson, Kim Whitehall, Rishi Verma, Lewis McGibbney, and Paul Ramirez. 2015. SciSpark: Applying in-memory Distributed Computing to Weather Event Detection and Tracking. In *IEEE BigData*. IEEE, 2020–2026.

[31] Stavros Papadopoulos, Kushal Datta, Samuel Madden, and Timothy Mattson. 2016. The TileDB Array Data Storage Manager. *Proceedings of the VLDB Endowment* 10, 4 (2016), 349–360.

[32] Nathalie Pettorelli. 2019. *Satellite remote sensing and the management of natural resources.* Oxford University Press.

[33] Kamleshan Pillay et al. 2014. Modelling changes in land cover patterns in Mtunzini, South Africa using satellite imagery. *Journal of the Indian Society of Remote Sensing* 42, 1 (2014), 51–60.

[34] PostGIS 2022. PostGIS. https://postgis.net/.

[35] QGIS Development Team. 2022. *QGIS Geographic Information System.* QGIS Association. https://www.qgis.org

[36] CA: Environmental Systems Research Institute Redlands. 2022. ArcGIS. https://www.esri.com/en-us/arcgis/about-arcgis/overview

[37] Elia Scudiero, Todd H Skaggs, and Dennis L Corwin. 2014. Regional scale soil salinity evaluation using Landsat 7, western San Joaquin Valley, California, USA. *Geoderma Regional* 2 (2014), 82–90.

[38] USDA National Agricultural Statistics Service. 2015. CropScape - Cropland Data Layer. (12 2015). https://doi.org/10.15482/USDA.ADC/1227096

[39] Andrii Shelestov, Mykola Lavreniuk, Nataliia Kussul, Alexei Novikov, and Sergii Skakun. 2017. Exploring Google Earth Engine platform for big data processing: Classification of multi-temporal satellite imagery for crop mapping. *frontiers in Earth Science* 5 (2017), 17.

[40] Samriddhi Singla, Ahmed Eldawy, Rami Alghamdi, and Mohamed F. Mokbel. 2019. Raptor: Large Scale Analysis of Big Raster and Vector Data. *Proc. VLDB Endow.* 12, 12 (2019), 1950–1953. https://doi.org/10.14778/3352063.3352107

[41] Samriddhi Singla, Ayan Mukhopadhyay, Michael Wilbur, Tina Diao, Vinayak Gajjewar, Ahmed Eldawy, Mykel Kochenderfer, Ross Shachter, and Abhishek Dubey. 2021. Wildfiredb: An open-source dataset connecting wildfire spread with relevant determinants. In *Conference on Neural Information Processing Systems Track on Datasets and Benchmarks*.

[42] Petr Sloup. 2016. GPU-accelerated raster map reprojection. *Geoinformatics FCE CTU* 15, 1 (2016), 61–68.

[43] Mathias Steinbach and Reinhard Hemmerling. 2012. Accelerating batch processing of spatial raster analysis using GPU. *Computers & Geosciences* 45 (2012), 212–220.

[44] Michael Stonebraker, Paul Brown, Donghui Zhang, and Jacek Becla. 2013. SciDB: A Database Management System for Applications with Complex Analytics. *Computing in Science and Engineering* 15, 3 (2013), 54–62.

[45] Michael Stonebraker, Paul Brown, Donghui Zhang, and Jacek Becla. 2013. SciDB: A Database Management System for Applications with Complex Analytics. *Computing in Science Engineering* 15 (05 2013), 54–62.

[46] U.S. Geological Survey. 2024. Landsat 8. https://www.usgs.gov/landsat-missions/landsat-8

[47] Planet Team. 2018–. Planet Application Program Interface: In Space for Life on Earth. https://api.planet.com

[48] Dejun Teng, Furqan Baig, Qiheng Sun, Jun Kong, and Fusheng Wang. 2021. IDEAL: a Vector-Raster Hybrid Model for Efficient Spatial Queries over Complex Polygons. In *2021 22nd IEEE International Conference on Mobile Data Management (MDM)*. 99–108. https://doi.org/10.1109/MDM52706.2021.00024

[49] TileDB, Inc. 2023. *tiledb: Universal Storage Engine for Sparse and Dense Multidimensional Arrays.* https://github.com/TileDB-Inc/TileDB-R R package version 0.20.1.

[50] Eleni Tzirita Zacharatou, Harish Doraiswamy, Anastasia Ailamaki, Claudio Silva, and Juliana Freire. 2017. GPU rasterization for real-time spatial aggregation over arbitrary polygons. *Proceedings of the VLDB Endowment* 11, 3 (2017).

[51] USGS 2024. Earth Observing Satellites. https://www.usgs.gov/calval/earth-observing-satellites.

[52] Randall T Whitman, Michael B Park, Sarah M Ambrose, and Erik G Hoel. 2014. Spatial indexing and analytics on Hadoop. In *Proceedings of the 22nd ACM SIGSPATIAL international conference on advances in geographic information systems*. 73–82.

[53] Ye Wu, Ying Ge, Weibao Yan, and Xinyu Li. 2007. Improving the performance of spatial raster analysis in GIS using GPU. In *Geoinformatics 2007: Geospatial Information Technology and Applications*, Vol. 6754. SPIE, 244–254.

[54] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. 2016. Simba: Efficient in-memory spatial analytics. In *Proceedings of the 2016 international conference on management of data*. 1071–1085.

[55] Chenghai Yang et al. 2012. Using high-resolution airborne and satellite imagery to assess crop growth and yield variability for precision agriculture. *Proc. IEEE* 101, 3 (2012), 582–592.

[56] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. 2015. GeoSpark: A cluster computing framework for processing large-scale spatial data. In *SIGSPATIAL*.

[57] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. 2016. A Demonstration of GeoSpark: A Cluster Computing Framework for Processing Big Spatial Data. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 1410–1413.

[58] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. 2019. Spatial Data Management in Apache Spark: The GeoSpark Perspective and Beyond. *GeoInformatica* 23 (2019).

[59] Ramon Antonio Rodriges Zalipynis. 2018. ChronosDB: Distributed, File based, Geospatial Array DBMS. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1247–1261.

[60] Jianting Zhang. 2011. Speeding up large-scale geospatial polygon rasterization on GPGPUs. In *Proceedings of the ACM SIGSPATIAL Second International Workshop on High Performance and Distributed Geographic Information Systems* (Chicago, Illinois) *(HPDGIS '11)*. Association for Computing Machinery, New York, NY, USA, 10–17.

[61] Jianting Zhang and Simin You. 2012. CudaGIS: report on the design and realization of a massive data parallel GIS on GPUs. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on GeoStreaming* (Redondo Beach, California) *(IWGS '12)*. Association for Computing Machinery, New York, NY, USA, 101–108.

[62] Jianting Zhang, Simin You, and Le Gruenwald. 2015. Large-scale spatial data processing on GPUs and GPU-accelerated clusters. *Sigspatial Special* 6, 3 (2015), 27–34.

[63] Liangpei Zhang, Lefei Zhang, and Bo Du. 2016. Deep learning for remote sensing data: A technical tutorial on the state of the art. *IEEE Geoscience and remote sensing magazine* 4, 2 (2016), 22–40.