Seamless Transition of Data Analyses and Analytics from a Local Workstation to Scalable, Massively Distributed Processing on the Cloud Using the Open Source PODPAC Library

Jerry Bieszczad[1], Mattheus Ueckermann[1], Dara Entekhabi[2], David Callender[1], and David Sullivan[1]

[1]Creare LLC
[2]Massachusetts Institute of Technology

November 24, 2022

Abstract

Newer satellite platforms, such as NISAR, are poised to produce huge amounts of data that require large computational resources. Currently, researchers typically download datasets for analysis on local computer resources. This paradigm is no longer practical given the volumes of data from new sensing platforms. While cloud computing services offer a potential solution for accessing and managing large computational resources, there remains a significant barrier to entry. Levering cloud services requires users to: navigate new terminology without appropriate documentation; optimize settings for services to reduce costs; and maintain software dependencies, upgrades, and allocated hardware resources. A more accessible approach for migrating earth scientists to the cloud is needed. To address this problem, we are developing the open source Python library PODPAC (Pipeline for Observational Data Processing Analysis and Collaboration), with the goal of helping to address NASA's rapidly growing observational data volume and variety needs. PODPAC enables earth scientists to seamlessly transition between processing on a local workstation (their current paradigm) to distributed remote processing on the cloud. It does this by leveraging a text-based JSON format automatically generated for any plug-and-play algorithm developed using PODPAC (e.g., in a Jupyter Notebook). This text format describes data provenance, and is used in RESTful web requests to preconfigured PODPAC cloud deployments, allowing scalable, massively distributed processing. We demonstrate the seamless transition to the cloud by developing a simplified soil moisture downscaling algorithm in Python using PODPAC. Data for this algorithm uses NASA Soil Moisture Active Passive (SMAP) sensor retrieved from the National Snow and Ice Data Center using OpenDAP, and fine-scale topographic data retrieved via Open Geospatial Consortium (OGC) Web Coverage Service (WCS) calls. We then use a serverless AWS Lambda function to run the same algorithm using the automatically-generated text format. Our generic preconfigured environment can handle a wide variety of processing pipelines, and scale up to 1024 parallel processes. This approach enables incremental adoption of cloud services by researchers, significantly lowering the barrier to entry.

IN41D-0858
AGU 100 FALL MEETING
Washington, D.C. | 10-14 Dec 2018

Seamless Transition of Data Analyses and Analytics from a Local Workstation to Scalable, Massively Distributed Processing on the Cloud Using the Open Source PODPAC Library

J. Bieszczad*, M. Ueckermann*, D. Entekhabi†, D. Callender*, and D. Sullivan* *Creare LLC, † Massachusetts Institute of Technology

Motivation

- NASA seeks to **migrate EOSDIS observational data products to the** Amazon Web Service (AWS) **cloud**
- AWS enables data scientists to **exploit massively scalable processing capabilities** on the cloud
- However, **barriers to entry are high**, given the unfamiliar terminology, setup, workflow, costs, etc. of AWS resources

amazon web services | IAM Roles | EC2 | Lambda | API Gateway | S3 | Glacier | Route 53

Project Objectives

- Develop PODPAC, an open-source, Python library, which **removes major barriers to widespread exploitation of EOSDIS and other earth science data** on the cloud
- Automate geo-data wrangling for **integrated analyses of disparate data sources** in a plug-and-play manner
- Enable data scientists to **easily transition workstation analyses to massively distributed processing** on AWS
- Facilitate **generation and sharing** of reproducible and documented earth science data products and algorithms

Open Source Development

- PODPAC is open-source software available at https://podpac.org
- **Wanted**: Testers, early adopters, contributors and feedback

Acknowledgment

- This research is supported by NASA under SBIR Phase II Contract No. 80NSSC18C0061

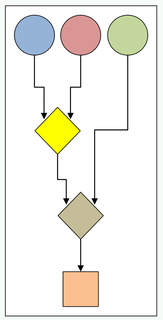
Pipeline for Observational Data Processing, Analysis, and Collaboration

Readily develop integrated geospatial analyses and analytics on your workstation

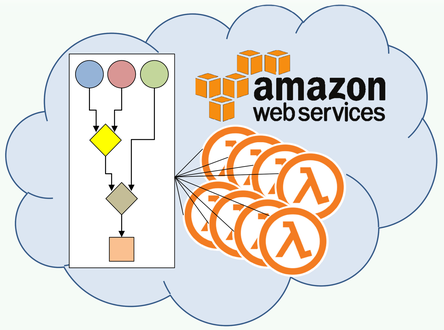
Encapsulated Local and Remote Data Sources

Plug-and-Play User Algorithms

Documented Earth Science Data Products

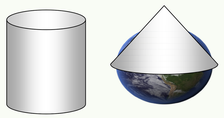


Seamlessly transition to scalable, massively distributed processing on the cloud



- ✓ Use of Python and Jupyter Notebooks reduces software learning curve for new users
- ✓ Local and remote data (OPeNDAP, WCS, etc.) are encapsulated in common API wrapper for plug-and-play integration within user-specified algorithms
- ✓ Automated data wrangling handles differences in geospatial CRS, projections, resolution, formats, etc.

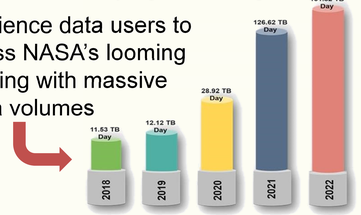
Geospatial CRS and Projections



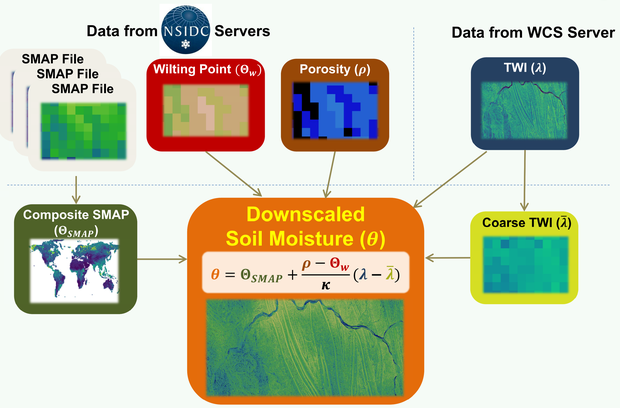
Data Structures

- ✓ Generated data products automatically record data provenance (sources, algorithms, versions) for reproducibility and documentation via JSON metadata

- ✓ JSON metadata enables direct deployment and execution of PODPAC algorithm pipelines on AWS
- ✓ PODPAC-enabled "serverless" AWS Lambda functions avoid provisioning and maintenance of cloud servers
- ✓ PODPAC Lambda functions automatically scale up to 1024 parallel computational processes
- ✓ Processing on AWS "close to data storage" improves performance and avoids costly egress charges
- ✓ Migrating earth science data users to AWS helps address NASA's looming challenges in dealing with massive earth science data volumes



"Serverless" SMAP Downscaling



SMAP Downscaling Using PODPAC in a Jupyter Notebook

Create PODPAC nodes for accessing NSIDC data via OPeNDAP

```
In [2]: import podpac  # Make the "PODPAC" library available to Python
        smap = podpac.datalib.smap.SMAP(interpolation='bilinear', cache_type='disk')
        wilt = podpac.datalib.smap.SMAPWilt(interpolation='bilinear', cache_type='disk')
        porosity = podpac.datalib.smap.SMAPPorosity(interpolation='bilinear', cache_type='disk')
```

Create PODPAC node to access topographic wetness index (TWI) from a WCS server

```
In [3]: twi = podpac.data.WCS(source=podpac.utils.load_setting('WCS_URL'),
                              layer_name=podpac.utils.load_setting('TWI'),
                              interpolation='nearest')
```

Reproject high resolution TWI onto low resolution SMAP grid

```
In [4]: twi_bar = podpac.data.ReprojectedSource(source=twi,
                                                reprojected_coordinates=smap.shared_coordinates,
                                                interpolation='bilinear')
```

Define downscaling algorithm

```
In [5]: downscaled_sm = podpac.algorithm.Arithmetic(A=smap, B=twi, C=twi_bar, D=porosity, E=wilt,
                                                    eqn='A + (D - E) / 13.0 * (B - C)')
```

Specify geospatial region and datetime of interests

```
In [6]: coordinates = podpac.Coordinates([podpac.clinspace(41., 40., 916),
                                          podpac.clinspace(-77, -76, 916),
                                          '2017-09-03T12:00:00'], dims=['lat', 'lon', 'time'])
```

Evaluate downscaling algorithm on local workstation

```
In [7]: downscaled_sm.eval(coordinates)
```

Evaluate downscaling algorithm via AWS Lambda functions

```
In [8]: lambda_node = podpac.core.managers.aws_lambda.Lambda(source=downscaled_sm)
        downscaled_soil_moisture = lambda_node.eval(coordinates)
```