Machine Learning Analytic Services in EDAS

Thomas Maxwell, Thomas Favata, Dan Duffy, Laura Carriere, Jerry Potter
Earth System Grid Federation
Compute Working Group

- ESGF distributes the data that supports the IPCC Assessment Reports
- CWT provides server-side analytics for ESGF
- CWT has defined a python API and a WPS service
- NASA-NCCS has implemented an ESGF-CWT analytics server (EDAS)
- Enables distributed, high-performance analytics close to the data

Estimated Data Growth of ESGF
2012 AR5 – 2 to 5 petabytes
2017 AR6 – 25 to 50 petabytes
2022 AR7 – 100 to 1,000 petabytes
NASA ESGF CWT Analytics (EDAS)

Public

ESGF - NASA ESGF CWT Analytics (EDAS)

DMZ

EDAS Web Service
ESGF-CWT-WPS

Apache Tomcat

WPS

INTERNAL

EDAS Analytic Services
DASK / XArray

0MQ

Data

0MQ

IBM GPFS

Python

NASA
EDAS Analytic Services Framework

- Implemented in 100% python
  - Access to full python analytics ecosystem
- Built on Dask/Xarray
  - Dask-distributed parallelism
- Restful WPS interface
  - Esgf-cwt compliant
- Workflow framework
  - Compose graphs of canonical operations
- Parallel data access
  - Directly from POSIX or OpenDAP
Dask-distributed parallelism

• **Familiar APIs:**
  - XArray builds on numpy and netCDF APIs.
  - High level constructs and automatic parallelism simplify development

• **Pure Python:**
  - Built in Python using well-known technologies

• **Large group of developers**

• **Low latency:**
  - Each task suffers about 1ms of overhead

• **Peer-to-peer data sharing:**
  - Workers communicate with each other to share data

• **Complex Scheduling:**
  - Supports complex workflows (not just map/filter/reduce)

• **Data Locality:**
  - Scheduling algorithms cleverly execute computations where data lives
Xarray Data Analysis Toolkit

- Extends Pandas to support N-dimensional arrays.
  - Inherits performance and power of Pandas.
- Tight integration with numpy and netCDF
  - In memory representation of netCDF data using np.ndarray
- Integrated with Dask for streaming data parallelism
  - Transparent distributed (chunked) arrays
  - Lazy, streaming computation on datasets that don’t fit in memory
  - Builtin parallel NetCDF IO
  - Automatically parallelizes xarray workflows
  - Parallelized numpy builtin and ufunc operations.
EDAS Architecture

Public

- Jupyter
- Python
- ESGF

DMZ

- EDAS Web Service
- ESGF-CWT-WPS
- Apache Tomcat

WPS

INTERNAL

- Jupyter
- Python
- EDAS Analytic Services
- DASK / XArray

0MQ

NASA

Data

IBM GPFS
Request Structure

domains = [{ name: d0, lat: {0, 50}, lon: {0, 100}, time: {‘1980-01-01’,’2005-01-01’} } ]
variables = [{ col: merra, name: tas,hur,va,ua, domain: d0, result: v0 }]
{ col: merra2, name: tas,hur,va,ua, domain: d0, result: v1 } ]
operations = [{ name: xarray.diff, input: v0,v1, result: vdiff }]
{ name: xarray.std, input: vdiff, axes: e } ]

Collection: MERRA
Variables: tas,hur
Input
Lat: { 0, 50 }
Lon: { 0, 100 }
Time: { 1980, 2005 }

Domain
Collection: MERRA2
Variables: tas,hur,va,ua

Input
Collection: MERRA2
Variables: tas,hur,va,ua

Difference

Input
Collection: MERRA
Variables: tas,hur,va,ua

Standard Deviation
Axis: e
Kernels

Canonical operations:

- Data access & subset
- Average (weighted and unweighted)
- Maximum
- Minimum
- Sum
- Difference
- Product
- Standard Deviation
- Variance
- Anomaly
- Median
- Norm
- Filter
- Decycle
- Highpass/Detrend
- Lowpass/Smooth

Specialized operations:

- EOF
- PC
- TeleconnectionMap

Neural Network Kernels:

- Layer
- Trainer
- Model
Canonical Operation Options

- **Domain**: subset to region of interest
- **Axes**: reduce over axes
  - X (latitude), Y (longitude), Z (levels), T (time), E (ensemble)
- **Groupby**: split-apply-combine
  - Custom or existing Axis
  - Pandas groups
- **Resample**: upsampling and downsampling
  - Pandas resample API

Example (for 10 years of data):

<table>
<thead>
<tr>
<th>Operation</th>
<th>Interpretation</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ave( axis: t )</td>
<td>Time average</td>
<td>1</td>
</tr>
<tr>
<td>ave( axis: te)</td>
<td>Time ensemble average</td>
<td>1</td>
</tr>
<tr>
<td>ave( axis: t, groupby: t.month)</td>
<td>Monthly climatology</td>
<td>12</td>
</tr>
<tr>
<td>ave( axis: t, resample: t.month)</td>
<td>Monthly means</td>
<td>120</td>
</tr>
</tbody>
</table>
class StdKernel(OpKernel):
    def __init__(self):
        OpKernel.__init__(self, KernelSpec("mean", "Standard Deviation Kernel",
            "Computes the standard deviation of the array elements",
            "along the given axes."))

    def processVariable(self, request: TaskRequest, node: OpNode, variable: EDASArray,
        attrs: Dict[...], products: List[...]) -> List[EDASArray]:
        return [variable.std(node.axes)]
class TeleconnectionKernel(OpKernel):
    def __init__(self):
        OpKernel.__init__(self, KernelSpec("telemap", "Teleconnection Kernel",
        "Produces teleconnection map by computing covariances at each point "
        "(in roi) with location specified by 'lat' and 'lon' parameters."))

    def processVariable(self, request: TaskRequest, node: OpNode, variable: EDASArray,
                        attrs: Dict[str, Any], products: List[str]) -> List[EDASArray]:
        parms = self.getParameters(node, [Param("lat"), Param("lon")])
        aIndex = variable.xr.get_axis_num('t')
        center: xa.DataArray = variable.selectPoint(float(parms["lat"]), float(parms["lon"])).xr
        cmean = center.mean(axis=aIndex)
        data_mean = variable.xr.mean(axis=aIndex)
        cstd = center.std(axis=aIndex)
        data_std = variable.xr.std(axis=aIndex)
        cov = np.sum((variable.xr-data_mean)*(center-cmean), axis=aIndex)/variable.xr.shape[aIndex]
        cor = cov / (cstd * data_std)
        return [EDASArray(variable.name, variable.domId, cor)]
Teleconnection Maps

Computes a map of covariances between a chosen point and all other points in the ROI.

<table>
<thead>
<tr>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lat: { 0, 80 }</td>
</tr>
<tr>
<td>Time: { 1980, 2010 }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection: CFSR</td>
</tr>
<tr>
<td>Variables: tas</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection: 20CR</td>
</tr>
<tr>
<td>Variables: tas</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection: ECMWF</td>
</tr>
<tr>
<td>Variables: tas</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection: MERRA2</td>
</tr>
<tr>
<td>Variables: tas</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis: t</td>
</tr>
<tr>
<td>Selection: djf</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>detrend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wsize: 15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>telemap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lat: 60</td>
</tr>
<tr>
<td>Lon: 310</td>
</tr>
</tbody>
</table>
EOF Workflow

EOF Decomposition:

\[ X(t, s) = \sum_{k=1}^{M} c_k(t) u_k(s), \]

- Domain
  - Lat: \{-80, 80\}

- Input
  - Collection: MERRA2
  - Variables: tas, hur, va, ua

- Decycle
  - Norm: True

- Detrend
  - Wsize: 5y

- Eofs
  - Nmodes = 4
MERRA2 Global Surface Temperature EOF Modes

Mode 0

Mode 1

Mode 2

Mode 3
MERRA2 Global Surface Temperature
Principal Component Timeseries

- **Mode = 0**
- **Mode = 1**
- **Mode = 2**
- **Mode = 3**
MERRA2 Global Surface Temperature
First EOF
PC1 – ENSO Index Comparison
Machine Learning Workflow

- Predict All-India Monsoon rainfall accumulation one year in advance
- Use a two-layer neural network
- Inputs: First 32 PCs of global surface temperature, 1 year lag time
Training Performance

- Loss Function: Mean square error
  - Output node results vs. IITM-AI timeseries
- Last 20% of data reserved for validation
- Choose model with minimum error on validation data
Applying the Neural Network Model

- Model kernel reads generated network structure and weights
- Generates a projection from a set of PCs
Results

- Comparison of predicted to actual monsoon precipitation
- Result of two month project by summer intern
Conclusions

• Big data analytics is moving closer to the data

• Workflows of canonical ops facilitate exploratory analytics

• Machine learning can exploit non-local climate dynamics