A System for Storing and Analyzing a Massive Climatological Database of Modeled Air Mass Trajectories

Daniel Pryden[1], Sean Raffuse[1], Narasimhan ("Sim") Larkin[2],
Thom Dedecko[1], Jennifer DeWinter[1], Kenneth Craig[1]

[1]Sonoma Technology, Inc.
Petaluma, California
[2]USDA Forest Service, Pacific Northwest Research Station
Seattle, Washington

Presented at the American Meteorological Society 92nd Annual Meeting,
Second Symposium on Advances in Modeling and Analysis Using Python
New Orleans, Louisiana
January 23, 2012

STi
Sonoma Technology, Inc.
Air Quality Research and Innovative Solutions

Arctic Black Carbon Study (ABCS)

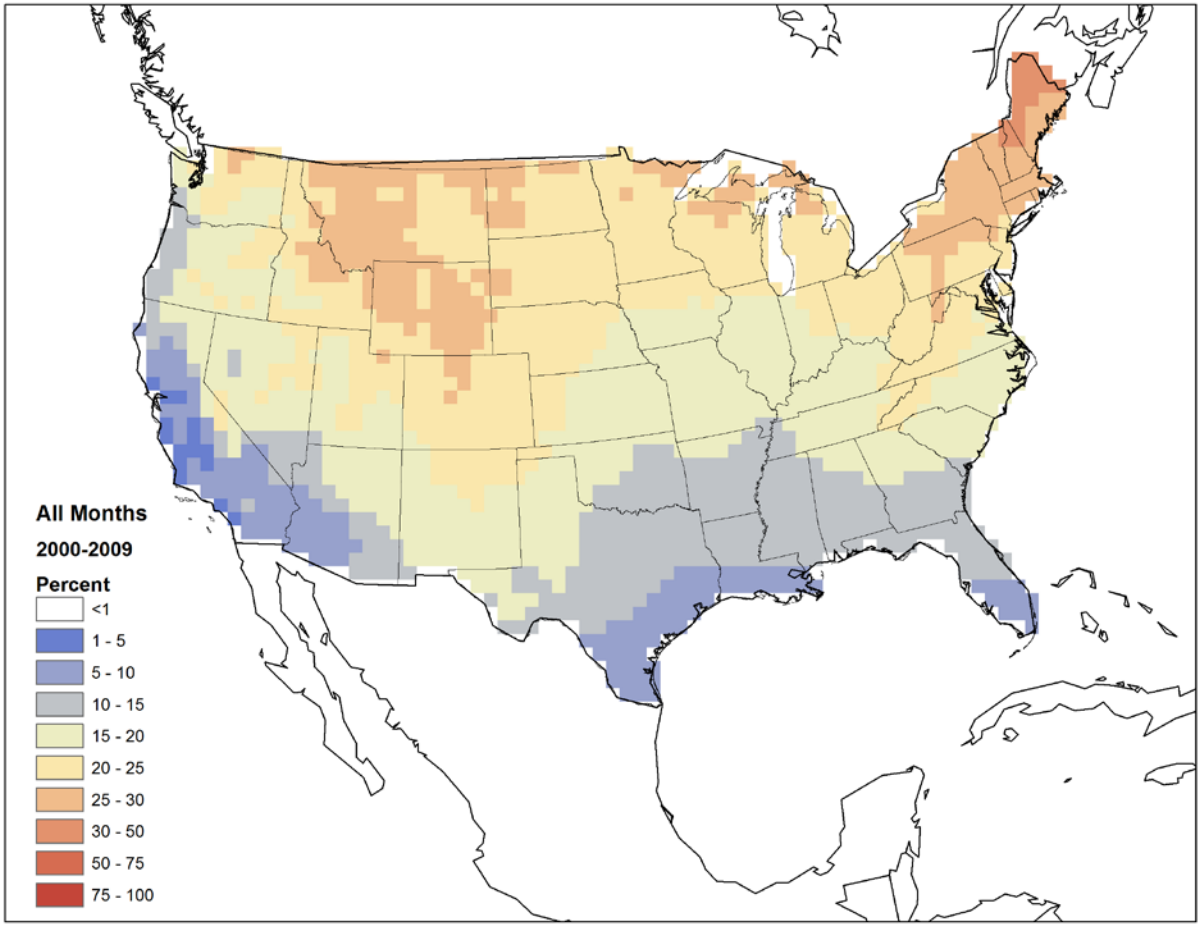
- Goal was to study patterns of transport of black carbon (BC) to the Arctic
- Can biomass burning (prescribed fire) affect climate change?
- Large-scale analysis using NOAA's HYSPLIT model
  - 30 years of data (1979-2008)

(Credit: NASA)

Large-Scale Data

- Modeled nearly **6×10^{8}** individual trajectories
- Final data set contained **1.4×10^{11}** points

- **How did we do it, and what did we learn?**



4.7×10^{10} data points went into this image.

Packed File Format for Trajectories (PFFT)

- HYSPLIT model produces results
  - Text format
  - Each waypoint on its own line
  - Each model initialization produces its own file
- We wanted
  - Compact binary format
  - Fast to read or write
  - Efficient for either full scan or random access
  - Multiple model initialization time steps in a file
- Solution: **Packed File Format for Trajectories (PFFT)**

Runner, Spawner, and Reaper

Goals

- Perform tens of thousands of separate runs of the HYSPLIT model
- Efficiently use many processors in parallel
- Collect results of many HYSPLIT runs into a single file

Modeled 30 years of trajectories in about 3 weeks with 24 cores

A three-part solution

- Runner.py
  - Command-line script; sets up date ranges and configuration
- Spawner.py
  - Launches `hymodelt` processes
  - When model executions finish, adds them to Reaper's work queue
- Reaper.py
  - Reads in HYSPLIT output and writes PFFT

How Do You Analyze 1.4×10^{11} Points?

- Only interested in aggregate results
  - At a minimum, aggregate per day
  - Usually, aggregates per month or year
- Want results on a map
  - Individual waypoints projected to grid cells
  - Value for a particular grid cell is an aggregate of values for all the trajectories that pass through that cell

Functional Programming

- Queries are implemented by defining functions
- Basic form is three functions: `filter()`, `map()`, and `reduce()`
- Processing takes the general form of a **left fold**, so aggregation queries can be performed in O(n) time with O(1) storage

```
filter_func = lambda traj: return True/False (predicate)
projection_func = lambda traj: return x, y, t
map_func = lambda traj: return object (initial grid-cell value)
reduce_func = lambda a, b: combine two grid-cell values
simplify_func = lambda value: return simplified value in correct type
```

Command-line Usage

- Through the magic of `eval()` we can define functions on the command line

```
make_grid.py
    -f FILTER_FUNC
    -m MAP_FUNC
    -r REDUCE_FUNC
    -s SIMPLIFY_FUNC
    -d DATA_TYPE
```

- Typical command line:

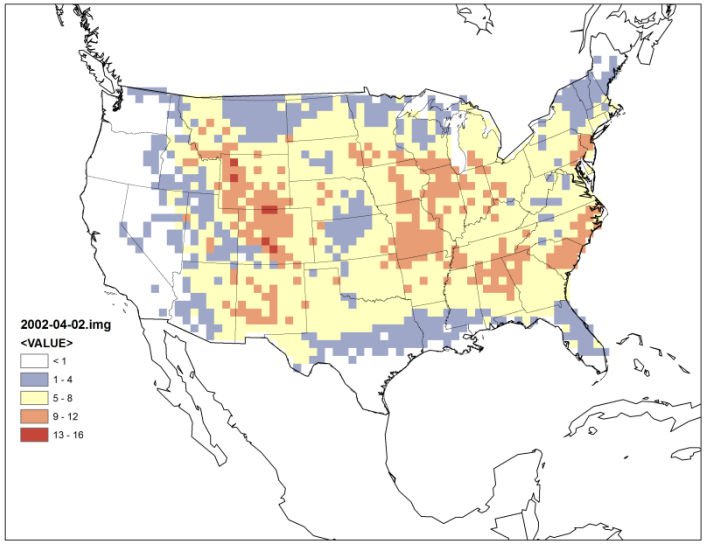
```
./make_grid.py -m "lambda traj: 1" -r "lambda a,
b: a + b" output_dir 2006010100 2006053118
/data/HYSPLIT/pfft/2006.pfft
```

Examples (1 of 2)

Count of trajectories per grid cell

```
# Here we count 1 for each trajectory in a grid cell,
# and we reduce by adding the counts together.

map_func = lambda traj: 1
reduce_func = lambda a, b: a + b
```



Examples (2 of 2)

Mean heights of trajectories

```
# This one is a bit more tricky.  We map each trajectory
# to a 2-tuple of (count_trajs, total_height).  Then the
# reduce_func can reduce them together by just adding
# the counts and the totals.  But we need a final step,
# in the simplify_func, to simplify our 2-tuple into an
# actual output value, by dividing the total by the count
# to come up with an actual average value.

map_func = lambda traj: (1, traj.height)
reduce_func = lambda a, b: (a[0] + b[0], a[1] + b[1])
simplify_func = lambda value: float(value[1]) / float(value[0])
```

Results

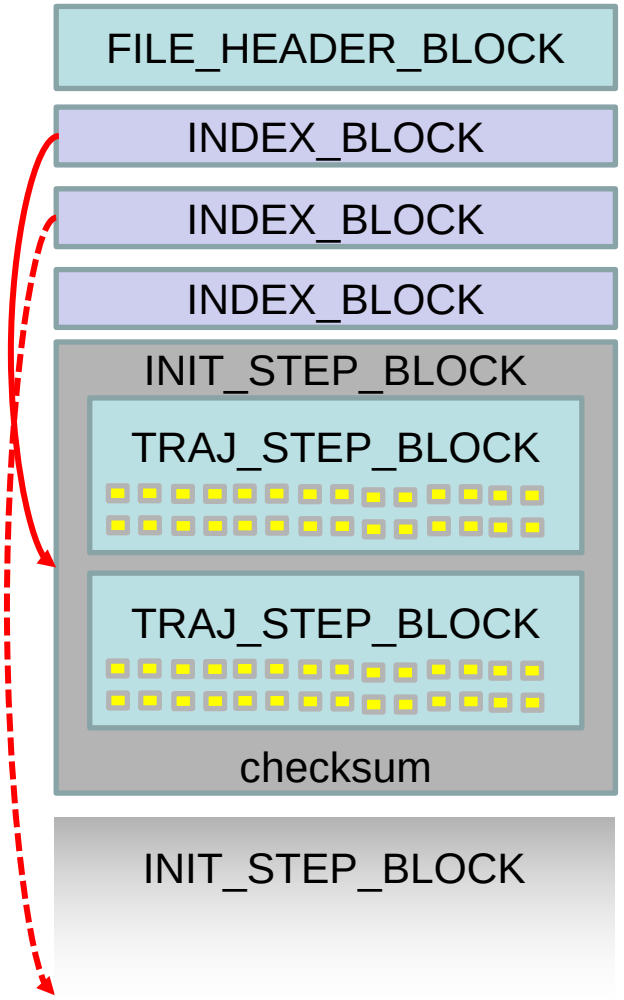
- What is the likelihood of transport from the United States to the Arctic? How does this likelihood vary regionally?
  - *Transport is possible (up to 42% in winter) at altitudes typical of prescribed fire injection (< 2,000 meters)*
  - *Transport is more likely at northern latitudes (varies regionally)*
- Is transport more likely during certain seasons or months?
  - *Transport is more likely during spring, fall, and winter*
  - *Seasonal, monthly, and daily variability due to synoptic patterns*

Conclusion

- Wrapping a command-line model with a Python class is a useful technique enabling large-scale use.
- Higher-order functions and functional programming techniques are a powerful way of constructing analyses on large data sets.

Additional Material: PFFT Structure

- Block sizes defined by header
- Values (latitude, longitude, height, etc.) stored as 32-bit floats
- Reserved spaces for future extensions
- Timestamps and file offsets are 64-bit
- Adler-32 checksums



Additional Material: PFFT Implementation

Implementation

- Uses Python's `struct` module to read and write values
- Adler-32 checksum implementation courtesy of the `zlib` module
- Lazy reading using generator functions (`yield` keyword)

Usage

```
import sys
from pfft import PfftFile

pfft = PfftFile(sys.argv[1])

print "  num_init_steps:", pfft.num_init_steps
print "  num_traj_steps:", pfft.num_traj_steps
print "  num_traj_points:", pfft.num_traj_points
print "  extra variables:", pfft.extra_variable_names

for init_step in pfft:
    print "InitStepBlock:",
    print init_step.initialization_time.strftime("%Y-%m-%d %H:%M")

    for traj_step in init_step:
        print "  TrajStepBlock",
        print traj_step.date_time.strftime("%Y-%m-%d %H:%M")
        for traj in traj_step:
            print "    %s: (%s, %s, %s)" % (
                traj.trajectory_number,
                traj.latitude,
                traj.longitude,
                traj.height,
                )
            for var in pfft.extra_variable_names:
                print "      %s: %s" % (var, traj[var])
```

Running HYSPLIT: hysplit.py

- Internal library used on several projects
- Wrapped command-line `hymodelt` executable using `subprocess` module
- `DataLocation` class and subclasses determine the locations of meteorological data to provide to HYSPLIT, given a desired date/time range

General Form (Pseudocode)

```
# Process through PFFT data
for init_step in pfft_data.iter_steps():
    for traj_step in init_step.iter_traj_steps():
        for traj in traj_step.iter_trajs():
            if filter_func(traj):
                x, y, t = projection_func(traj)
                existing_value = data_cube[x, y, t]
                value = map_func(traj)
                if existing_value is not None:
                    value = reduce_func(existing_value, value)
                data_cube[x, y, t] = value

# Process data cube objects into numeric values
variable = new NetCDFVariable(x_size, y_size, t_size) # etc.
for x in range(data_cube.x_size):
    for y in range(data_cube.y_size):
        for t in range(data_cube.t_size):
            variable[x, y, t] = simplify_func(data_cube[x, y, t])
```