DSG with netCDF Enhanced Data Model

2017 EarthCube netCDF-CF Workshop

Challenges

- Irregular point data are more difficult to map to netCDF data model than grids
- Duplicate information...
- Or need ragged arrays

Requirements

- Work well with existing clients
  - Python
  - netCDF-Java
- Support realtime data streams
- My use cases:
  - U.S. upper air data
  - U.S. surface station (i.e. METAR)

Contiguous Array

```
netcdf stations {
  dimensions:
    obs = UNLIMITED;       // currently 6
  variables:
    float lat(obs);
    float lon(obs);
    string stid(obs);
    double time(obs);
    float temperature(obs);
  data:
    lat = 39.9,40,35.25,39.9,35.25,39.9;
    lon = -104.9,-105,-97.1,-104.9,-97.1,-104.9;
    stid = "KDEN","KBOU","KOKC","KDEN","KOKC","KDEN";
    time = 7776000,7776000,7776000,7777800,7777800,
           7778100;
    temperature = 15,16,25,15.5,25.2,15.6;
}
```

Contiguous Array

- One dimension for observations
- Every array has the same size
- Repeats values for spatial metadata
- Simple
- Need to scan entire file to find all observations for a particular station
- In NetCDF-3, wastes space
- Compression makes this less worrisome (?)

Ragged Array

```
netcdf stations {
  dimensions:
    obs = UNLIMITED;        // currently 6
    station = UNLIMITED;    // currently 3
  variables:
    float lat(station);
    float lon(station);
    string stid(station);
    int index(obs);
    double time(obs);
    float temperature(obs);
  data:
    lat = 39.9,40,35.25;
    lon = -104.9,-105,-97.1;
    stid = "KDEN","KBOU","KOKC";
    index = 0,1,2,0,2,0;
    time = 7776000,7776000,7776000,7777800,7777800, 7778100;
    temperature = 15,16,25,15.5,25.2,15.6;
}
```

Ragged Array

- One dimension for observations, one for stations
- Extra observation-length array for index mapping observation to station
- Less extra space usage
- More complicated
- Need to scan entire file to find all observations for a particular station
- Less redundant information

Extended Model

- Can the netCDF extended model make these simpler?
  - Compound types
  - Vlens
  - Groups

Compound Types

- Does not address ragged array challenge
- Performance optimization

Compound Data Type, Obs Dim

```
netcdf stations {
  types:
    compound obs_t {
      float lat;
      float lon;
      string stid;
      double time;
      float temperature;
    }
  dimensions:
    obs = UNLIMITED;      // currently 6
  variables:
    obs_t obs(obs);
  data:
    obs = {39.9,-104.9,"KDEN",7776000,15},
          {40,-105,"KBOU",7776000,16},
          {35.25,-97.1,"KOKC",7776000,25},
          {39.9,-104.9,"KDEN",7777800,15.5},
          {35.25,-97.1,"KOKC",7777800,25.2},
          {39.9,-104.9,"KDEN",7778100,15.6};
}
```

Compound Data Type, Obs Dim

- One dimension for observations
- Compound data type used to keep all values for a particular observation
- Better data locality may improve I/O, compression
- Dealing with compound data type may be complex for client
- Same repeated information as Contiguous Array--but could be addressed
- This CDL crashed ncgen (fixed)

Vlen

- Variable length **data type**
- ~~Solves the ragged array problem~~
- What would really help is Vlen as a dimension--as a data type it incurs some “challenges”

VLen Data Type, Station Dim

```
netcdf stations {
  types:
    float (*) variable_time_float;
    double (*) variable_time_double;
  dimensions:
    station = UNLIMITED;      // currently 3
  variables:
    float lat(station);
    float lon(station);
    string stid(station);
    variable_time_double time(station);
    variable_time_float temperature(station);
  data:
    lat = 39.9,40,35.25;
    lon = -104.9,-105,-97.1;
    stid = "KDEN","KBOU","KOKC";
    time = {7776000,7777800,7778100}, {7776000},
           {7776000,7777800};
    temperature = {15,15.5,15.6},{16},{25,25.2};
}
```

VLen Data Type, Station Dim

- One dimension for observations
- VLen data type used to handle ragged arrays
- Finding data for a station only requires finding a single index
- All data for a particular variable stored together
- Similar to Indexed Ragged Array, but exploiting VLen to rationalize per-station storage

Groups

- Handle changing dimensionality by separating stations into their own groups
- Essentially uses groups as a dimension

Group-per-station

```
netcdf stations {
  dimensions:
    station = UNLIMITED ;
  variables:
    float lat(station) ;
    float lon(station) ;
    string stid(station) ;
  data:
    lat = 39.9, 40, 35.25 ;
    lon = -104.9, -105, -97.1 ;
    stid = "KDEN", "KBOU", "KOKC" ;
  group: KDEN {
    dimensions:
      obs = UNLIMITED ;
    variables:
      int index ;
      double time(obs) ;
      float temperature(obs) ;
    data:
      index = 0 ;
      time = 7776000, 7777800,
             7778100 ;
      temperature = 15, 15.5, 15.6;
  } // group KDEN
  group: KBOU {
    dimensions:
      obs = UNLIMITED ;
    variables:
      int index ;
      double time(obs) ;
      float temperature(obs) ;
    data:
      index = 1 ;
      time = 7776000 ;
      temperature = 16 ;
  } // group KBOU
```

Group-per-station

- All station metadata stored at global scope
- Each station has its own group
- Eliminates ragged arrays, VLen, and compound data types
- Client must scan all groups to get data for all stations
- No need for all stations to have same variables
- Can clients handle ~6000 groups?
- Locality probably same as using compound data type

All Together now

```
netcdf stations {
  types:
    compound obs_t {
      double time;
      float temperature;
    }
    obs_t (*) variable_obs_t
    compound stn_t {
      float lat;
      float lon;
      string stid;
      variable_obs_t obs;
    }
  dimensions:
    station = UNLIMITED;
  variables:
    stn_t stations(station)
```

```
data:
    stations = {39.9,-104.9,"KDEN",
                {{7776000,15},
                 {7777800,15.5},
                 {7778100,15.6}}},
      {40,-105,"KBOU",
       {{7776000,16}}},
      {35.25,-97.1,"KOKC",
       {{7776000,25},{7777800,25.2}}}
}
```

All Together Now

- “Compound VLen Data Type nested within Compound Data Type, Unlimited Station Dimension”
- Most complex
- Not clear all clients can even handle this
- Puts all information for a station together
- Slicing across stations is hard

Benchmarks

- Used collection of 400k METAR records
- Ragged arrays 25% faster to write and 35% smaller
- No performance benchmarks for new stuff

Take Aways

- All of the extended model extensions require manual iteration
- In Python, this has made testing and development excruciating
- Vlen is problematic specifically for realtime data--you don’t append values

My Opinion?

- Vlen is not worth the complexity
- Compound data types are not as bad...but I don't see the benefits either
- Groups are good, though I have concerns about having to scan through 6000 of them
- Without a true ragged array (i.e. vlen ***dimension***) classic model is fine