**nabu: A distributed, parallel, data processing platform**

Antonio T. Lorenzo*, William F. Holmgren*

College of Optical Sciences*, Department of Hydrology & Atmospheric Sciences*, University of Arizona

### Introduction

nabu is a tool designed to process data in parallel on distributed machines using Python. It will be used to generate operational solar and wind power forecast for electric utility companies in the Southwest US. These forecasts cover five to seven days in the future requiring a blending of short-term, statistical forecasts with long-term, NWPP forecasts, and forecasts will be regenerated every five minutes. The current forecast system relies on a large, monolithic python program that consumes 10 GB of memory and may take five minutes to produce forecasts (and plots). nabu will consume a fraction of the memory and wall time while also being easier to develop and debug.

### Design

- **Written in a functional style to improve reproducibility and make the program easier to understand.**
- **Uses Dask.distributed which generates an optimized execution graph that can be executed on workers on many different machines with minimal modification to a serial code.**
- **Utilizes Dask graphs to store provenance information for how a given forecast was generated.**
- **Relies on a REST data API that enables access to raw data from nearly any host (an NFS server/mount would also work).**

### Goals

- **Generate blended power forecasts for ~50 solar power plants along with aggregates for forecasts horizons from 5 minutes to 7 days.**
- **Update forecasts every 5 minutes.**
- **Leverage multi-core CPUs and a GPU to produce forecasts quickly.**
- **Make the forecast generation code easy to understand, extend, and debug.**
- **Make the forecast generation chain reproducible.**

### Simplified Example

A simplified example of how nabu processes data is shown to the right. We define functions that get the data from the REST API, compute a solar power forecast from irradiance and weather data, compute a persistence forecast, simulate a satellite image forecast, and combine the forecasts together before storing the result. Each function is called in the blue box, but the @delayed ensures no computation is performed until the client compute command. This command generates the computation graph shown below and sends it to the Dask.distributed scheduler. The scheduler assigns the tasks (nodes on the graph) to other worker processes intelligently to reduce data transfers between workers. We also show that a worker with access to a GPU should be assigned the task to generate satellite forecasts.

```
from dask.distributed import delayed
from dask.distributed import Client

# satellite forecast is
# First connect to the scheduler that's already running
# Now go ahead and run everything on our already setup
# occurs yet

# Get satellite data

# Get WRF forecasts

# Make the combined forecast

# Post the optimally combined forecast to the central API
```

### Helpful Tips

- **Make sure NetCDF4/HDF5 files are chunked in the shape that best fits your access pattern. In our case, compression and re-chunking reduced the time it takes to retrieve a point forecast from a WRF NetCDF from almost a minute to tens of milliseconds.**
- **The msgpack format with blosc compression is a fast serialization type for many python objects.**

---

*Note: the image is a diagram illustrating the workflow and architecture of the nabu system. The diagram includes several components such as Host Server, REST Data API (Flask), Dask.distributed Workers, Dask.distributed Scheduler, GPU Server, Output, and Utility Partners. It also shows the interactions between these components, including GET/POST operations and data flows through MySQL, GET/observations, GET/twr, and more. The diagram is not fully transcribed due to the nature of the image.*