Massively Parallel Analysis System for Seismology (MsPASS)

Prof. Gary L. Pavlis
Department of Earth and Atmospheric Sciences
Indiana University, Bloomington, IN

Overview

- Presentation on what MsPASS is and why you should use it
- Hands on session working through jupyter notebook

MsPASS

1. Massively Parallel
   - Single CPU thing of the past
   - Archaic software infrastructure

2. Analysis System for Seismology
   - Research analysis – flexibility
   - For Seismology research – tool for us not computer scientists

https://www.mspass.org/

Discussion

- MsPASS is a framework for research computing. What does the "research" keyword mean in this context?

Production versus Research IT systems

Production

- Solves Specific Problem
- Performance is critical
  - Time is money
  - Mission critical role with repeating tasks
- Operable with minimum skill set necessary for job
- Data model well known and fixed

Research

- Handle range of problems
- Performance is secondary
  - Feasible sufficient
  - Many one-up solutions
- Can assume users are specialists and life-long learners
- Data highly variable (one person's signal is another's noise)

Key Roles MsPASS can play in research today

- Manage large data sets
- Tools for low-level work to assemble event-based data
- Generic parallel tools for handling continuous data
- Complete set of tools for P-wave receiver function processing
- Tools for spectral estimation

Hands on exercise focus

What is MsPASS good for

- Generic waveform processing toolkit
- Scalability
  - Prototype on a desktop
  - If needed scale up to HPC or cloud
  - Likely to become the tool of choice for interacting with Earthscope DMC when they get their cloud system working
- With new Earthscope system your MsPASS notebook will allow reproducing at least the low-level preprocessing to assemble your dataset

What is in the box?

- Exceptionally complete documentation (https://mspass.org) for open-source package
- Database to manage large data sets
- Parallel processing
  - HPC or cloud functionality
  - Clean error handling for long running jobs
- Flexible but simple to use IO abstractions (read-write almost any format and from URL)
- Python job control language
  - Reproducibility automatic if you submit jupyter notebook with your paper
- Algorithms – low level preprocessing (solved problem) focus
  - All obspy signal processing algorithms
  - Low-level windowing, bundling, and similar grungy stuff
  - Three-component primitives (rotation and transformation)
  - Trace editng
  - Header math
  - SNR calculations
  - All common P receiver function deconvolution+novel new method
- Docker container or anaconda package distribution

Sermonette/Sales Pitch

- This community needs to recognize what we have given you
- MsPASS makes archaic
  - SAC
  - AH
  - Most, if not all, existing generic waveform handling tools
- MsPASS supercedes ObsPy
  - If you use ObsPy you can use MsPASS with minimal effort
  - Drastically increases functionality over ObsPy

I know ObsPy well, why should I move to MsPASS?

- If you are only working with 10000 or less waveforms at a time it won't help
- For large data sets
  - Integrated database for data management essential
  - Clean error handling to run large jobs running for hours
  - Parallel processing functionality
  - Cross-platform compability – runs on desktop to highest end supercomputer or cloud system
- Likely to become the standard for low-level interaction with new Earthscope cloud system
- Reproducibility

Documentation, documentation, documentation:

- User Manual
- Python API Reference
- C++ API
- Tutorials – you all should have cloned this repository already

MsPASS is NOT

- Ideal solution for problems we viewed as solved:
  - Real time data handling
  - Seismic event catalog processing
  - Seismic reflection processing
  - Archival data management for Earthscope
- A complete solution for all of seismology
  - This is a framework, not a turnkey solution to all problems – may require some work to develop your custom “workflow”
  - More custom solutions require contributions from people like you

Hands on exercise part one: Data management

- Indexing miniseed files
- Managing source metadata
- Managing receiver (station) metadata
- Seismic data reader and writer abstraction

MsPASS Data Management Technology

- MongoDB is the integrated database of MsPASS
- MongoDB is a “document database”
  - Jargon name: a document == python dictionary
  - A “document” is the ultimate seismic header
    - You can save almost anything there
    - Completely flexible (by default) in what you save
- See the Data Management section of the MsPASS User Manual

To Jupyter Notebook

- Launch docker container as described here
- Probably will want to append “:v2.0.1” tag to run line

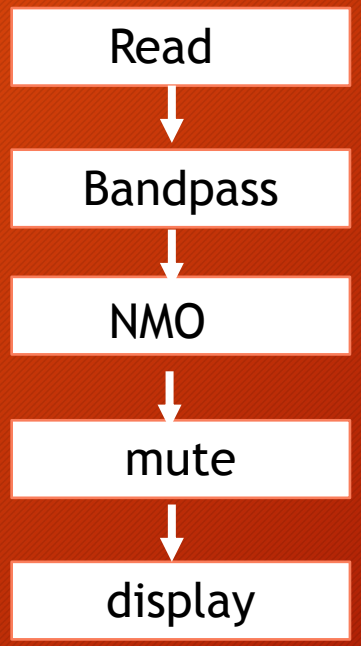
Part 2: Parallel schedulers

- Map-reduce model
  - Modern jargon term
  - Online sources obscure some simple concepts
  - Main paradigm for MsPASS parallel processing
- A modern paradigm likely to have a long lifetime
- I will introduce by analogy to unix pipeline concepts I assume all of you know

Seismic Reflection Workflow model

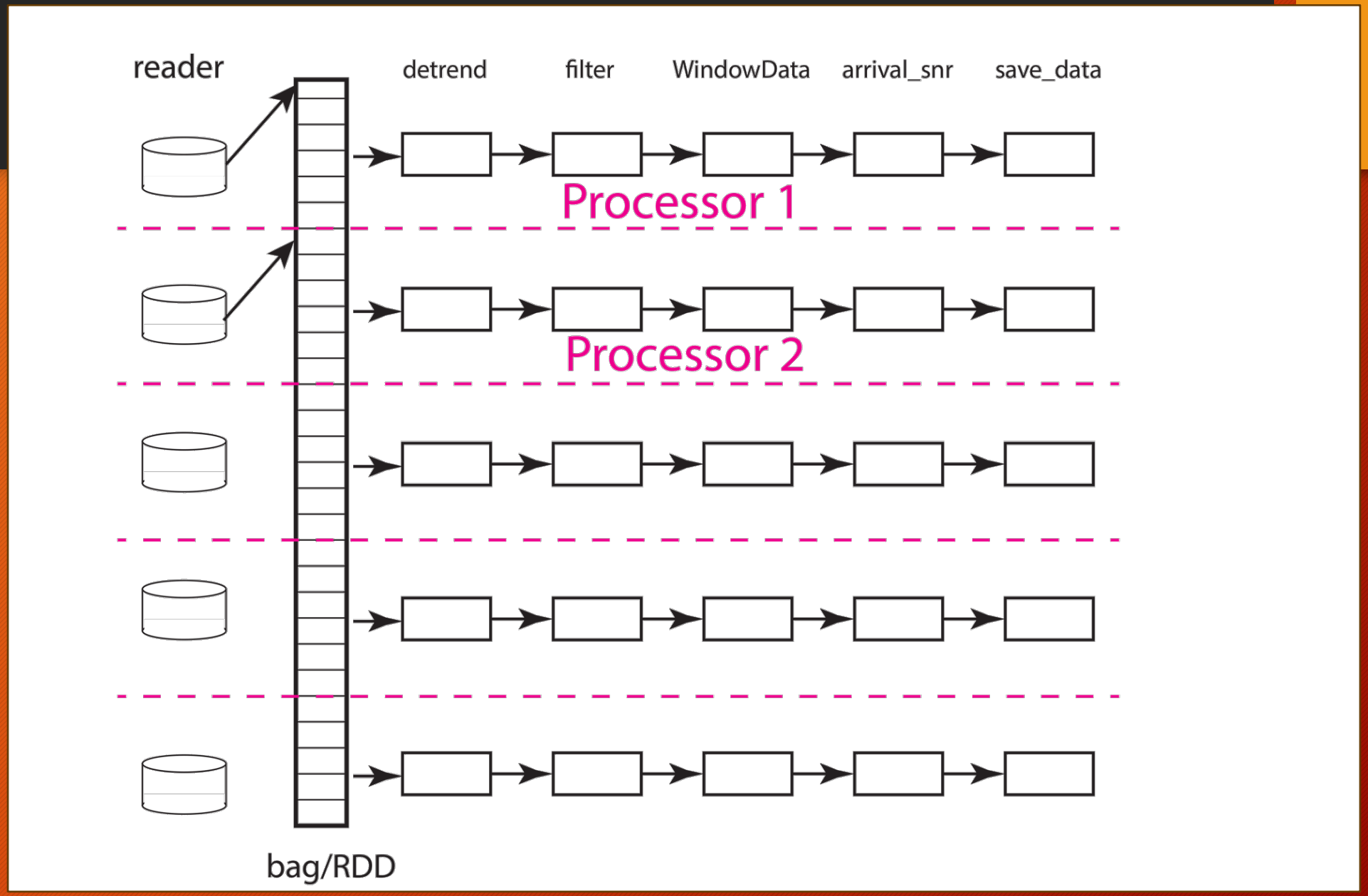
Example: seismic unix

- Illustrates traditional model
- Key point is data flow through processors
- Processors read input, modify it, and emit output

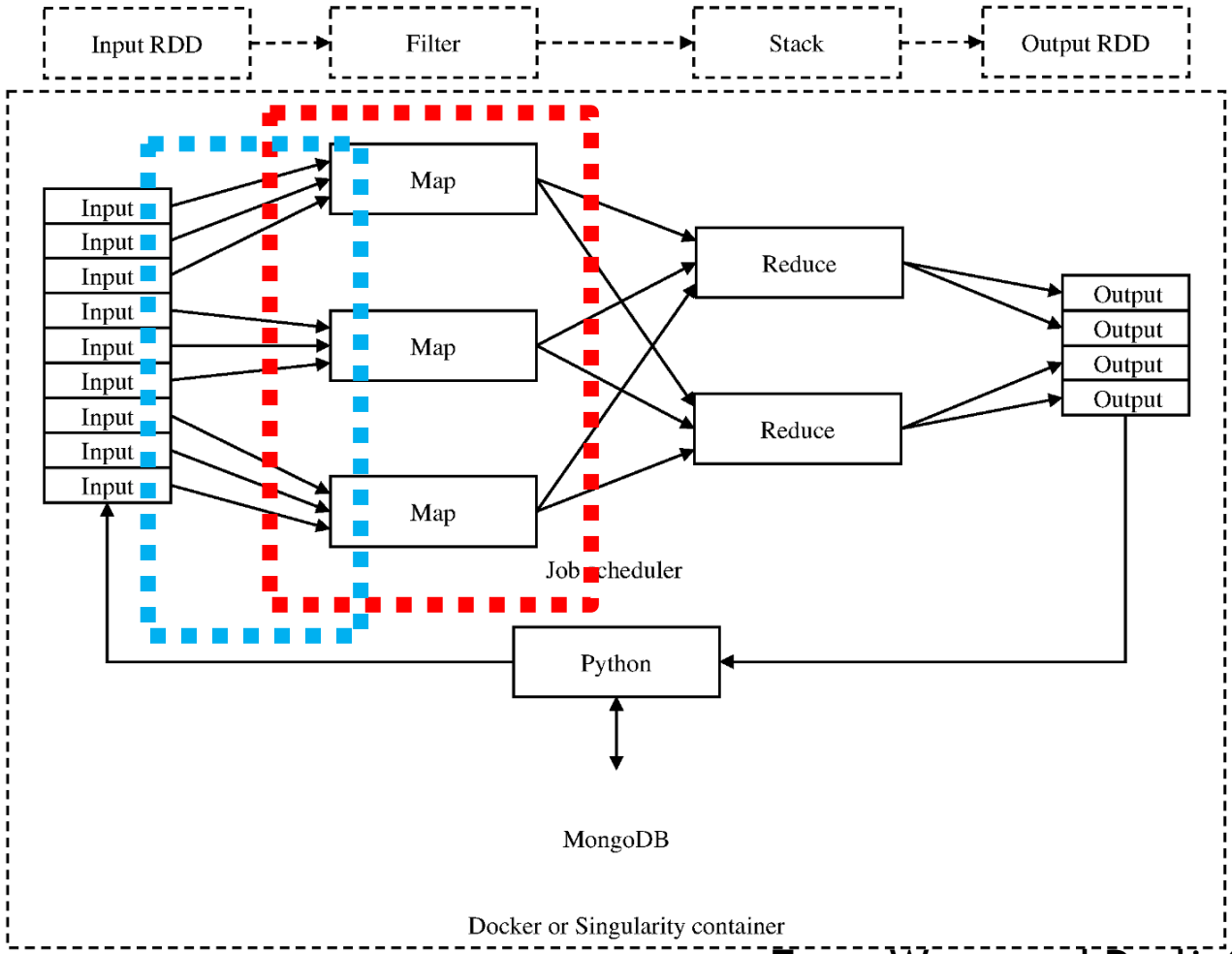



```
# real examples would add arguments for parameters
subfilt < mydata | sunmo | sumute | suxwig
```

How to parallelize a similar job (2 processors)



MsPASS Parallelization: map-reduce model

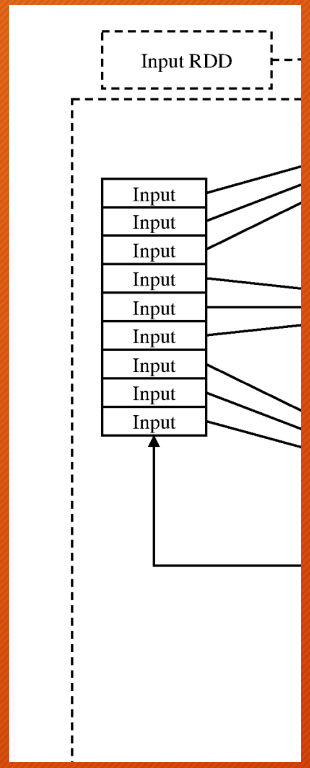


From Wang and Pavlis(2021)

Key points

- Map operator
  - Behaves like a unix filter
  - Scheduler assigns each datum to processes
- Reduce operator
- Data flow
  - Parallel pipeline
  - Conceptually similar to unix shell | symbol
  - Scheduler moves data
  - Faster for threads than nodes

Spark RDD == Dask bag



- Documentation for both Spark and Dask obscure this topic
- RDD/bag concepts
  - Algorithmically identical to a large array of things (objects)
  - Workers can pull any component in equal time
  - The collection of things (data set) may not fit in memory
- Other parallel containers
  - Dataframe (table)
  - Array (matrix bigger than memory)

Comparison of serial and parallel workflows

Serial

```
cursor = db.wf_Seismogram.find({})
for doc in cursor:
  d = db.read_data(doc,collection="wf_Seismogram")
  d = signals.detrend(d,'demean')
  d = signals.filter(d,"bandbass",
    freqmin=0.01,freqmax=2.0)
  d = WindowData(d,200.0,500.0,t0shift=d.t0)
  db.save_data(d,collection="wf_Seismogram",
     data_tag="results")
```

Parallel (Dask)

```
cursor = db.wf_Seismogram.find({})
data = read_distributed_data(db, cursor)
data = data.map(signals.detrend,'demean')
data = data.map(signals.filter,"bandpass",
freqmin=0.01, freqmax=2.0)
# windowing is relative to start time.  300 s window
starting at d.t0+200
data = data.map(lambda d : WindowData(d, 200.0,
500.0, t0shift=d.t0))
res = data.map(db.save_data,
     collection="wf_Seismogram",data_tag="results")
data_out = data.compute()
```

Key point: loop processing easily translated to series of map operators

Result acts like: detrend < datafile | filter > outfile