HPC Day: Advanced Topics (Part 2)

Presented By: Joseph Guzman

Date: 2025-09-19

Link to slides:

http://rcdata.nau.edu/hpcpub/workshops/advanced_topics_p1.pdf

NAU NORTHERN ARIZONA UNIVERSITY

Overview

- 1. Parallelism - Slurm Arrays and MPI Programming
- 2. Globus and File Management

Section 1: Parallelism - Slurm Arrays and MPI Programming

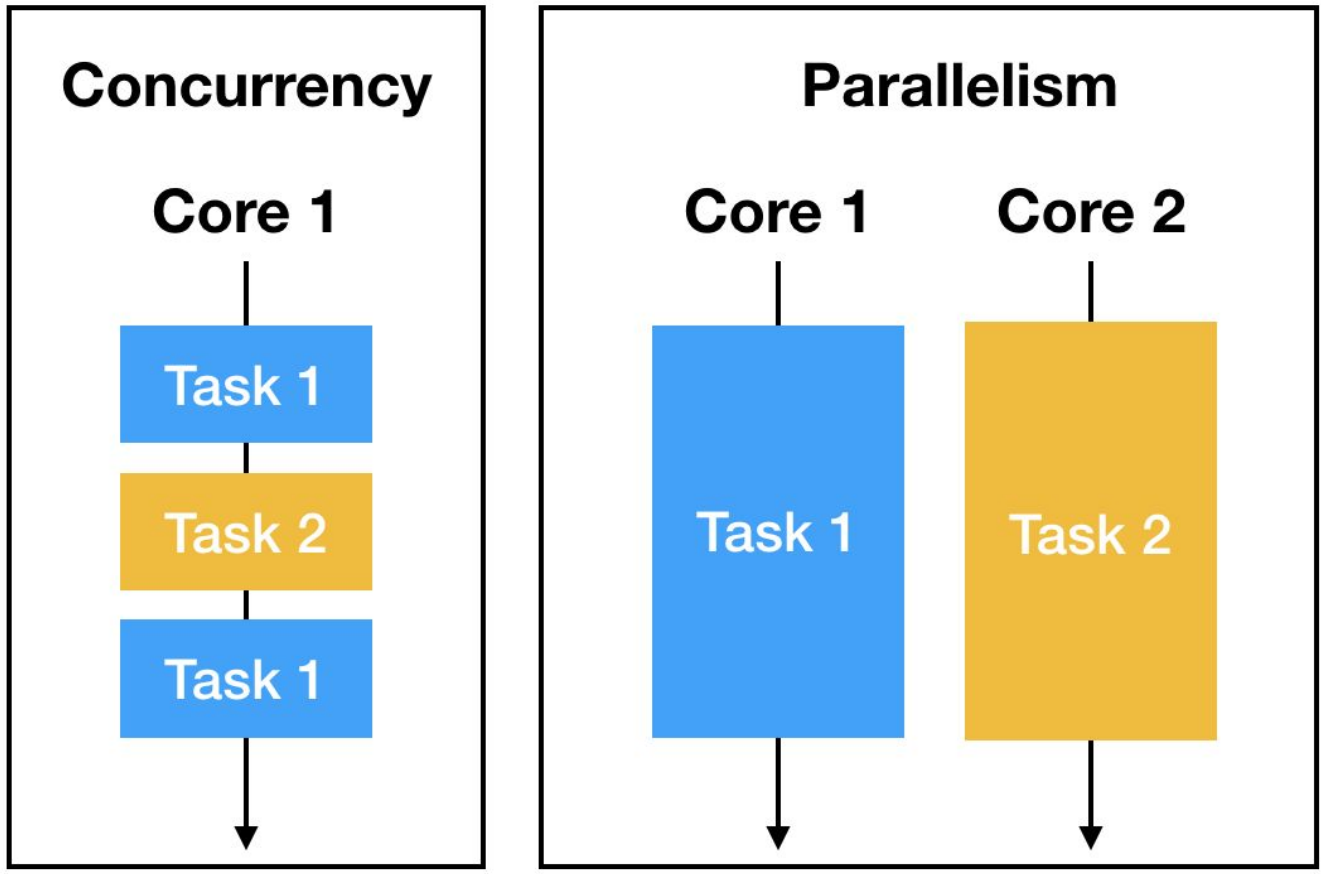
Types of Parallelism

- Each of your jobs can be categorized as:
  - Serial
  - Parallel (which we can describe as being one of these three types)
    - 1. Shared Memory
      - Independent logical tasks are able to share memory
        - Meaning you can have a C pointer to that object without any additional load ops
      - Heuristic: Runs on a single node
    - 2. Distributed Memory
      - Independent logical tasks are not able to share memory
      - Heuristic: Runs on multiple nodes
    - 3. Hybrid
      - Uses a mix of both

Methods of Parallelism

- Serial + Multi-Processing
  - Tools: Slurm Arrays, pipes, sockets, py multiprocessing module, etc.
- Shared Memory
  - Multithreading, Shared Memory (inter-process)
    - Tools: POSIX Thread API (pthreads C library)
  - (inter-process) Shared Memory
    - Tools: POSIX Shared Memory API, OpenMPI
- Distributed Memory
  - MPI (technically hybrid, but primarily distributed)
    - Tools: OpenMPI
  - Other network-enabled models (i.e. Client-Server, P2P, Load Balancing, etc.)
    - While these methods can enable parallelism, these models do not cater to HPC like MPI does
- Hybrid Memory
  - Any combination of shared+distributed memory tools

Concurrency vs. Parallelism



- You can have concurrency without parallelism.
- Using a parallel programming tool, does not mean that your job is parallel.

Why use parallelism?

- *Your code likely is not taking full advantage of the HPC environment if it's not using multiple threads or processes*
  - Monsoon does have high capacity file storage, RAM availability, GPUs, and network throughput.
  - But your serial computations will not happen any faster than they would on your desktop.
- The primary goal of this presentation is to expose slurm arrays and MPI to everyone.
  - These are tools that cater specifically to HPC environments.
- Note: if you're making use of high-level code, your packages/libraries may be parallelized in the backend (i.e. tensorflow-distributed). You should still be aware of how that works.

What is a Slurm Job Array? (part 1)

- Slurm Job Arrays can be the easiest way to parallelize non-parallel code
  - If you find yourself executing a repetitive set of commands, or an otherwise repetitive task
  - Then it's likely that job arrays can help you
- A job array works like a loop
  - you can request a number of identical job allocations all at once
  - where you execute a different task in each job
- Job arrays simplify accounting later, and will allocate quicker
  - A single jobid can identify the X number of job array elements with squeue/sacct/jobstats
    - Each job array element has a different suffix in the jobid as well
  - Executing thousands of individual sbatch commands can bog down the cluster, so we prefer that researchers use job arrays where possible

What is a Slurm Job Array? (part 2)

1. Job script is created

analysis
--array=8

Slurm Arrays

2. Job script is submitted

analysis
--array=8

3. Job is launched with eight instances running in parallel

analysis 123456_1	analysis 123456_2	analysis 123456_3	analysis 123456_4
analysis 123456_5	analysis 123456_6	analysis 123456_7	analysis 123456_8

Advantages:

- Submit one job script to create many jobs
- Only need to keep track of one jobid

Useful environment variables

SLURM_ARRAY_JOB_ID: the job array's ID (parent)

SLURM_ARRAY_TASK_ID: the id of the job array member n (child)

%A

%a

Creating a Slurm Job Array

- You need to use sbatch, we always recommend setting SBATCH parameters in a script file
- Components:
  - `#SBATCH --array=x-y`
    - replace ‘x’ and ‘y’ with an integer range (i.e. 1-10)
    - you can also have a comma-separated list
  - `SLURM_ARRAY_TASK_ID`
    - this is an environment variable that slurm sets when your job is executing
    - this environment variable will always be an integer that represents the index of the job array element
  - (optional) `#SBATCH --output=/path/to/my/output/jobname_%a.txt`
    - `%a` is a meta string you can use in the filename argument with the `--output` or the `--error` parameter that evaluates to the job index
    - `%A` works similarly, but for whole job arrays

Example: Counting Occurrences

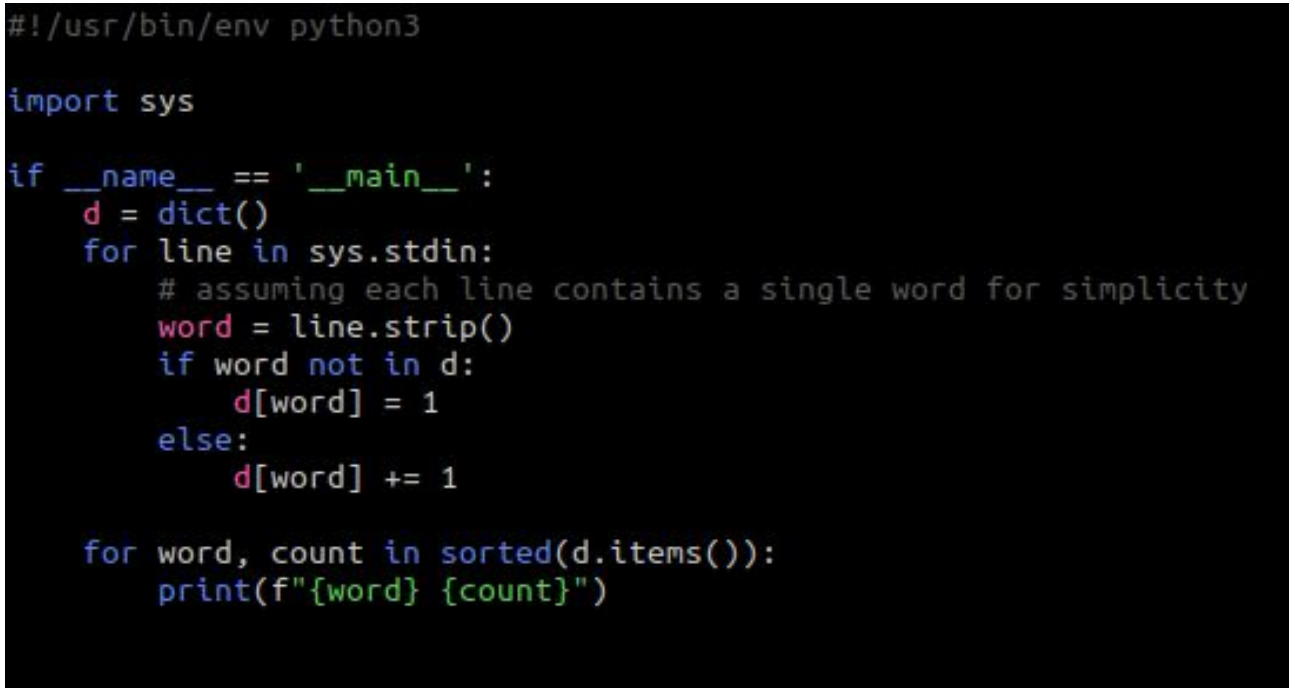
- One of the classic ***embarrassingly parallel*** problem is counting occurrences.
  - ***embarrassingly parallel*** or ***perfectly parallel*** describes a computing problem that requires little to no communication between parallel tasks.
    - Communication and especially synchronization is the hard part of parallel programming!
    - Some practical examples of ***embarrassingly parallel*** problems?
      - BLAST queries
      - Monte Carlo simulations
      - Bruteforce Hashing
      - ML Training Algorithms
    - We will be covering these two example problems:
      - Counting Occurences
      - Sorting

Counting Occurrences - Data

- All programs and data for this file are located here: /packages/workshop/in-depth
- Load the module with this command: `module load workshop/in-depth`
- Input Data:
  - 10 input files
    - Each containing 100 million random words, one word per line
    - filepaths:
      - /packages/workshop/in-depth/data/rand-words-1.txt
      - …
      - /packages/workshop/in-depth/data/rand-words-10.txt
- Sample:

```
$ head /packages/workshop/in-depth/data/rand-words-1.txt
nonvibratile
antinomy
yearners
fructified
donsy
malnutrite
oniony
reutilizations
preauditory
acetylphenylhydrazine
$
```

Counting Occurrences - Serial Implementation



```python
#!/usr/bin/env python3

import sys

if __name__ == '__main__':
    d = dict()
    for line in sys.stdin:
        # assuming each line contains a single word for simplicity
        word = line.strip()
        if word not in d:
            d[word] = 1
        else:
            d[word] += 1

    for word, count in sorted(d.items()):
        print(f"{word} {count}")
```

- count-words.py
- reads from stdin and populates a dictionary that holds the count for each unique line
- behavior is similar to this GNU coreutils command: `uniq -c`

Counting Occurrences - Serial Execution

- (optional) make a new directory to store your work
  - i.e.:
    - `mkdir /scratch/$USER/workshop`
      - The next example has large result files, use /scratch to avoid filling up your /home!
    - `cd /scratch/$USER/workshop`
- Load the workshop/in-depth module
  - `module load workshop/in-depth`
- Submit the job:
  - `sbatch $JOBSCRIPT_DIR/cw-serial.sh`
    - Note: `JOBSCRIPT_DIR` is an env var set when you load the module, all jobscripts discussed in this workshop have a home there.

- Runtime: 13 minutes
  - Pretty slow!
  - But let's use job arrays to speed it up!

Counting Occurrences - Divide and Conquer

- Divide and Conquer is the general strategy to use for most parallelizable problems.
  - Divide a large problem into smaller problems
  - Process those smaller problems in parallel
  - Combine results

Counting Occurrences - Job Array (main script)

```
#!/bin/bash

#SBATCH --job-name=cw-array
#SBATCH --array=1-10
#SBATCH --time=5:00

module load workshop/in-depth

basedir="/packages/workshop/in-depth"

cd "$basedir" || {
    echo "Error: could not cd into '$basedir'"
    exit 1
}

srun bash -c "count-words < ./data/rand-words-$SLURM_ARRAY_TASK_ID.txt \
> $OLDPWD/results-$SLURM_ARRAY_TASK_ID.txt"
```

- cw-array.sh
- Critical pieces:
  - where I specify the `--array` sbatch parameter
  - usage of the `SLURM_ARRAY_TASK_ID` environment variable

Counting Occurrences - Job Array (aggregates results)

```
#!/bin/bash

#SBATCH --job-name=cw-array-aggregate
#SBATCH --time=1

module load workshop/in-depth

srun bash -c "cat ./results-{1..10}.txt | \
awk '{ a[\$1]+=\$2 } END { for(i in a) print i, a[i] }' | \
sort -k 1 > results-aggregated.txt"
```

- cw-array-aggregate.sh
- Aggregates results by combining the results of each job array element

- **Submit this script with a dependency (optional):**
  - sbatch -d afterok:<jobid> $JOBSCRIPT_DIR/cw-array-aggregate.sh
    - replace <jobid> with the jobid of the main job array
    - Why? -- You cannot aggregate the results before that job completes, it will error out
- Or you could submit normally, simply monitor your job to make sure its done first

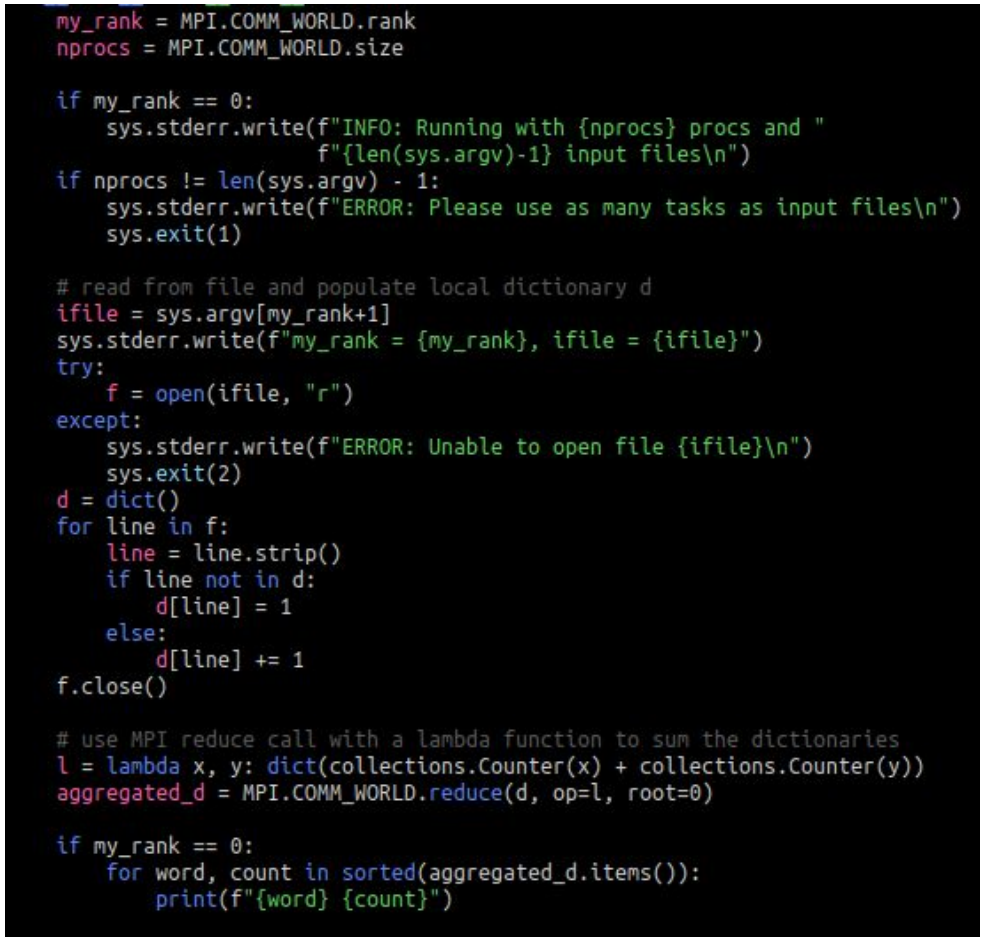
Counting Occurrences - Slurm Dependency Aside

- Slurm jobs launched via sbatch/srun/salloc can all make use of dependencies
- When a job has a dependency assigned to it, this means the job cannot start until a condition is met
  - Often this can be useful for pipelining
    - if you have a step2 that should not run before step1 is completed for example
      - as is the case in our example
  - Run man sbatch to read the details on how this works.
  - Some useful examples:
    - `-d afterok:<jobid>`
      - do not start job until specified <jobid> completes
    - `-d aftercorr:<jobid>`
      - do not start job array element, until corresponding job array element in <jobid> completes
    - `-d singleton`
      - do not start job until another job with the same name completes
        - useful for recurring jobs

Counting Occurrences - Job Array Execution

- `sbatch ./cw-array.sh`
  - record the jobid that sbatch produces, edit cw-array.sh to use this value
- `sbatch -d afterok:<jobid> ./cw-array-aggregate.sh`
  - this script will create the final output file: ./results-aggregated.txt
  - verify that this produces the same results as before (lines may be in a different order):
    - `diff results-serial.txt results-aggregated.txt`
- No edit to main count-words python script?
  - Correct, all I did to speed up the wallclock time was split it into parallel jobs
- Execution Time?
  - under ~1.5 minutes
    - that's the wallclock time, the aggregate CPU time is roughly equal

Counting Occurrences - MPI Python Script



```
my_rank = MPI.COMM_WORLD.rank
nprocs = MPI.COMM_WORLD.size

if my_rank == 0:
    sys.stderr.write(f"INFO: Running with {nprocs} procs and "
                     f"{len(sys.argv)-1} input files\n")
if nprocs != len(sys.argv) - 1:
    sys.stderr.write(f"ERROR: Please use as many tasks as input files\n")
    sys.exit(1)

# read from file and populate local dictionary d
ifile = sys.argv[my_rank+1]
sys.stderr.write(f"my_rank = {my_rank}, ifile = {ifile}")
try:
    f = open(ifile, "r")
except:
    sys.stderr.write(f"ERROR: Unable to open file {ifile}\n")
    sys.exit(2)
d = dict()
for line in f:
    line = line.strip()
    if line not in d:
        d[line] = 1
    else:
        d[line] += 1
f.close()

# use MPI reduce call with a lambda function to sum the dictionaries
l = lambda x, y: dict(collections.Counter(x) + collections.Counter(y))
aggregated_d = MPI.COMM_WORLD.reduce(d, op=l, root=0)

if my_rank == 0:
    for word, count in sorted(aggregated_d.items()):
        print(f"{word} {count}")
```

- count-words-mpi
  - A python script using mpi4py
  - It's a reimplementation of count-words, that's made parallel with MPI
  - **only the main body of the program is shown to the left**
- Not as trivial, but be glad it wasn't in C++!
- The crucial part is the last if statement
  - the ranks range from 0-9
  - all ranks, aside from rank 0, send their dictionaries to rank 0 with an MPI send operation
  - rank 0 receives them with an MPI receive operation, and prints the results to stdout

Counting Occurrences - MPI Python Job Script


```
#!/bin/bash

#SBATCH --job-name=cw-mpi
#SBATCH -n 10 # tasks may exist on different nodes
#SBATCH --time=5:00

module load workshop/in-depth

basedir="/packages/workshop/in-depth"

cd "$basedir" || {
    echo "Error: could not cd into '$basedir'"
    exit 1
}

srun bash -c "count-words-mpi ./data/rand-words-{1..10}.txt > \
$OLDPWD/results-mpi.txt"
```

- Job Script:
  - cw-mpi.sh
- Almost identical to the original cw-serial.sh
  - But notice the use of -n 10 here, this specifies 10 tasks for MPI

Counting Occurrences - MPI Execution

- sbatch ./cw-mpi.sh
- Main difference between this example and the job array?
  - We use MPI to aggregate results
  - Only slightly better computationally in this example
- Execution Time?
  - ~1.3 minutes
    - that's the wallclock time, the aggregate CPU time is roughly equal

Section 2: Globus and File Management

NAU NORTHERN ARIZONA UNIVERSITY

How do I move data to or from Monsoon?

Service	Tools	To Monsoon	From Monsoon
SSH	rclone, scp, rsync	YES	YES
SMB (a.k.a. samba, CIFS, “network drive”)	File Explorer, Other desktop client	YES	YES
FTP/FTPS	rclone, ftp	YES	NO
HTTP/HTTPS (Data Portal)	browser, rclone, libcurl, etc	NO	YES
Cloud (i.e. google drive)	browser, rclone, API, globus (NAU gdrive only)	YES	YES
Globus	browser (asynchronous)	YES	YES

- Other?
  - You can use Monsoon as a client for other services. But above are the service types that we support. Monsoon acts as a server for all of the services listed, except for “Cloud” and FTP/FTPS.

OpenSSH and Creating SSH Keys

- Highly recommended to use an OpenSSH Client
  - Will provide the OpenSSH implementation of ssh, scp, etc.
  - On Mac, Linux, and *BSD this should be installed by default
  - On Windows most people would recommend PuTTY, but you may also be able to get the Unix-centric OpenSSH client on new versions of windows with this command (see the MS docs here):
    - `Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0`
- To enhance security and convenience you can create an SSH key on your local machine.
  - `ssh-keygen -t ed25519`
- Then to use this key to log into monsoon you'd use this command:
  - ssh-copy-id abc123@monsoon.hpc.nau.edu
  - Now if you were to try to ssh into monsoon normally it will default to trying your key first, instead of prompting you.
  - ssh abc123@monsoon.hpc.nau.edu

SSH-Based Tools and SMB

SSH

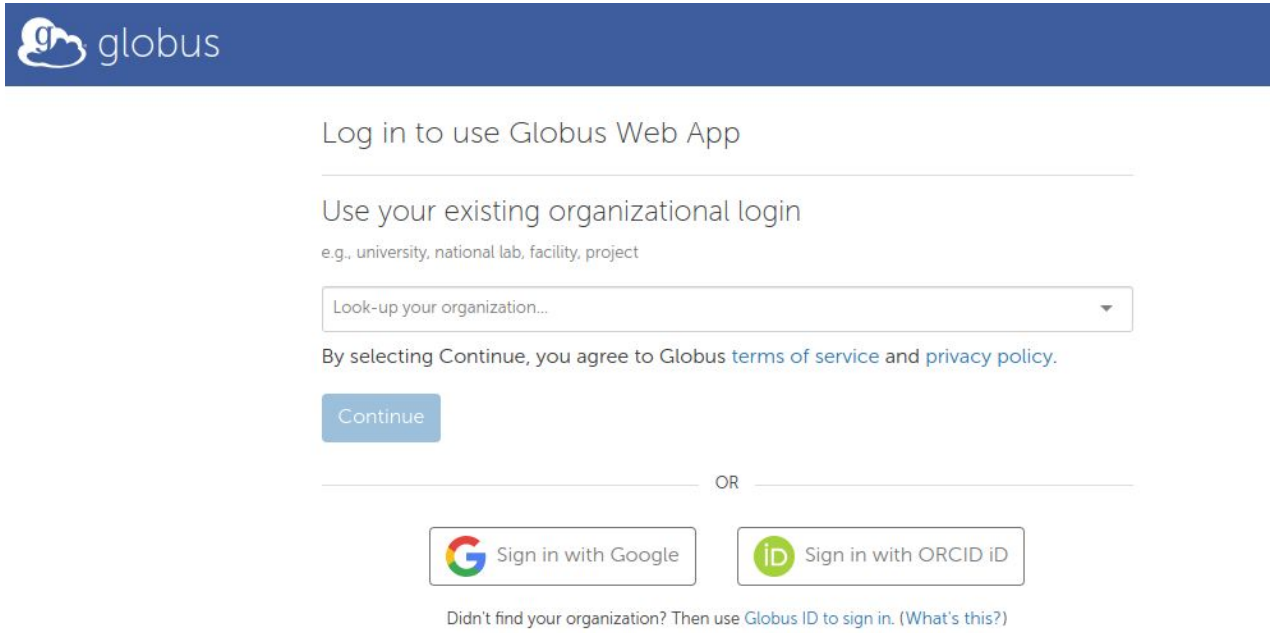
- scp
  - most simple, convenient for small downloads
- rsync
  - used for “syncing” a source and destination folder
    - performs better when you need to do partial downloads
- rclone
  - generalist tool, has support for many different service types
    - created primarily as a one-stop shop for interacting with cloud storage services
  - requires light configuration before using

SMB:

- Ideal for desktop usage
- Mount a network drive on your file explorer app
- Windows: \\shares.hpc.nau.edu\cirrus
- Mac: smb://shares.hpc.nau.edu/cirrus

Globus (Part 1)

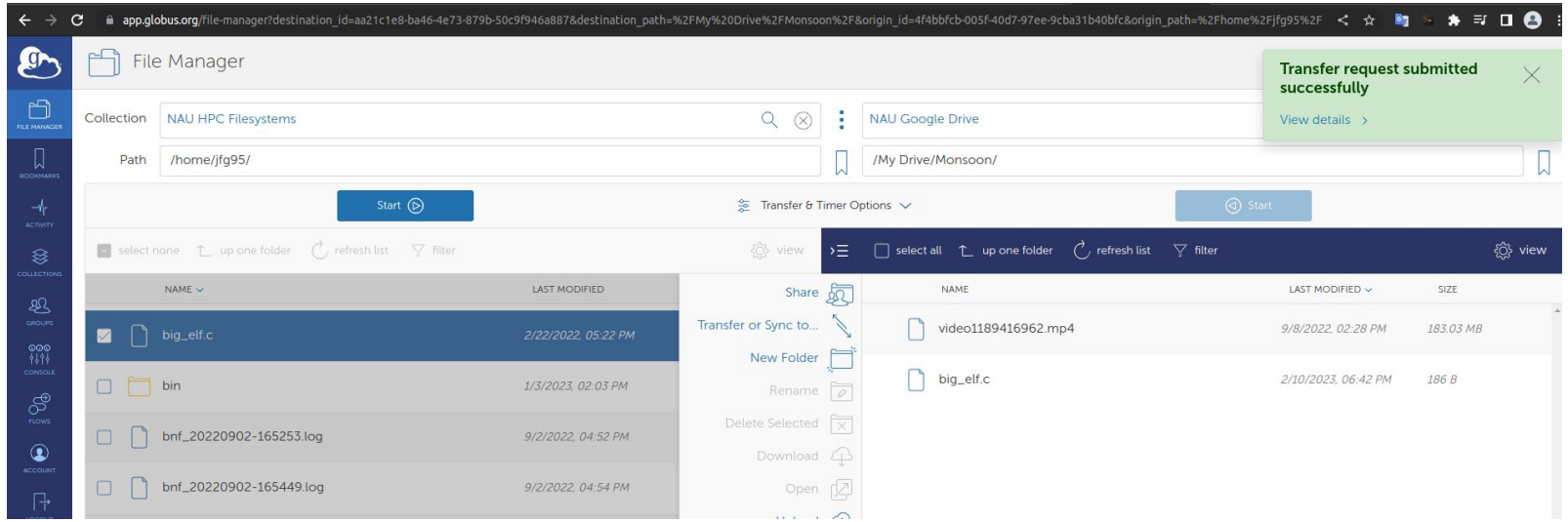
- Specialized HPC webapp accessible at https://app.globus.org
  - Primarily used by research institutions
  - Use-case: moving large amounts of data across institutions
  - Need the globus personal app to move data to or from your desktop
    - see: https://www.globus.org/globus-connect-personal)
- To start, visit the webapp and lookup “Northern Arizona University” at the prompt



- The first time it will ask you:
  - if you want to link your account
    - say no if you’ve never used globus before
  - to confirm your email
  - to accept terms and conditions
  - to allow globus to use your id for file transfers
- Uses NAU SSO

Globus (Part 2)

- Globus calls access points “collections”
- Search for “NAU” under “collections” to access Monsoon via globus
  - The one named “NAU HPC Filesystems” has access to /home /scratch and /projects
  - The one named “NAU Google Drive” has access to your NAU-affiliated google drive
  - You can read/write from your desktop too if you have globus personal running



- In the browser app
  - search up a collection
  - right click on a file or directory
  - select “transfer or sync to…”
  - now search a second collection
  - select a destination folder for the second collection
  - click on start to move data to the second collection

END

Need to contact us?

- Email:
  - ask-arc@nau.edu
- Zoom Office Hours
  - Every Wednesday
  - Alternates Between 1:00PM-2:00PM and 2:00PM-3:00PM
  - https://in.nau.edu/arc/office-hours/

Questions?

NAU NORTHERN ARIZONA UNIVERSITY

Section Header/Footer

ALT TITLE

NAU NORTHERN ARIZONA UNIVERSITY